

This electronic thesis or dissertation has been downloaded from the King's Research Portal at <https://kclpure.kcl.ac.uk/portal/>



Exploring Defenses Against Adversarial Attacks in Machine Learning-Based Malware Detection

Rashid, Aqib

Awarding institution:
King's College London

The copyright of this thesis rests with the author and no quotation from it or information derived from it may be published without proper acknowledgement.

END USER LICENCE AGREEMENT



Unless another licence is stated on the immediately following page this work is licensed

under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International

licence. <https://creativecommons.org/licenses/by-nc-nd/4.0/>

You are free to copy, distribute and transmit the work

Under the following conditions:

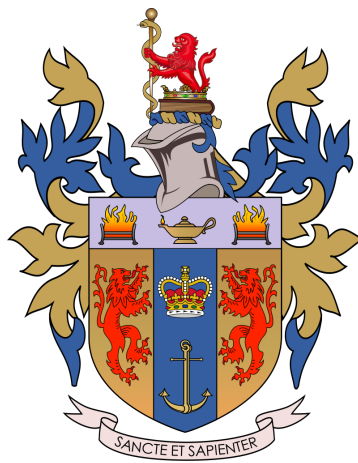
- Attribution: You must attribute the work in the manner specified by the author (but not in any way that suggests that they endorse you or your use of the work).
- Non Commercial: You may not use this work for commercial purposes.
- No Derivative Works - You may not alter, transform, or build upon this work.

Any of these conditions can be waived if you receive permission from the author. Your fair dealings and other rights are in no way affected by the above.

Take down policy

If you believe that this document breaches copyright please contact librarypure@kcl.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

Exploring Defenses Against Adversarial Attacks in Machine Learning-Based Malware Detection



Aqib Rashid

Department of Informatics
King's College London

This dissertation is submitted for the degree of
Doctor of Philosophy

2023

For their immeasurable love and support, this dissertation is dedicated to my family.

DECLARATION

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text.

Aqib Rashid
2023

ACKNOWLEDGEMENTS

In the name of Allah, the Gracious, the Merciful. With the sheer grace of God, I have made it here. I am eternally grateful that I have been given such great opportunities in this life, Alhamdulillah. “Make the most of your life, and get to the top”, I have always been encouraged.

During my Master’s and my PhD, I have had the utmost privilege of working with Professor Jose Such. His experience, intelligence, scientific insight, supervision, mentoring, and patience have been invaluable. He has always created a positive and inspiring environment that has helped me pursue particularly challenging aspects of research. I am extremely grateful to have had the honor and privilege of working with him.

At King’s, I met many other colleagues, BSc, MSc, and PhD students, and academics. Whether you taught me or I taught you, whether it was just a pleasant conversation while waiting for a lecture or the establishment of a lifetime of friendship, I express my sincere gratitude to you all. I am grateful to have been part of the supportive and friendly community at King’s. I am grateful for the funding received by the EPSRC.

I would also like to extend my sincere gratitude to Professor Chris Hankin and Associate Professor Jorge Blasco Alís, who served as my external examiners.

Prior to King’s, I was fortunate enough to be taught by many phenomenal teachers. I am eternally grateful to them all for dedicating their lives to ensure that my peers and I could have better futures. It is impossible to list them all here; know that you are remembered and will always be valued highly.

Over the years, I have made many great friends. I will always appreciate the positivity, friendship, and love that exists between us all. I look forward to sharing in our happiness and successes going forward, Insh’Allah.

Additionally, I am a member of the Ahmadiyya Muslim Community, whose spiritual leader is Hazrat Mirza Masroor Ahmad. I have had the great fortune of meeting his holiness many times, and I correspond with him regularly. Without his prayers and inspiration, all of this would have been impossible.

I would also like to thank all my family. It is impossible to describe in words how much I appreciate the prayers, love, affection, encouragement, and inspiration they have given me. To all my family, thank you.

The quote above is attributed to my beloved parents — Saqib, my father, and Sadia, my mother — who are a precious gift for someone like me. They have supported me in every possible way. I do not know how or if I can ever repay them; I will forever cherish them as parents and as friends. They have only ever wanted the best for me, and this work is dedicated to them. My brothers, Atif and Rayyan, have been my best friends from the beginning, and I will forever be thankful to have siblings like them. They are all true gems. Thank you to all of you.

Last but certainly not least, I want to mention a very special person, my fiancée, Laiba. Thank you very much for your support and encouragement.

ABSTRACT

Machine learning (ML) has facilitated progress in several disciplines as a result of greater resources, data volumes, and algorithmic developments. Particularly in cybersecurity, ML-based malware detection has proven to be an effective method of classifying software executables as benign or malicious. However, an unintended consequence of the increased usage of machine learning is that the attack surface has been widened through adversarial ML attacks. With such attacks, malicious users can craft input samples with the intention of evading ML models and causing them to output a specific prediction. For example, in an evasion attack, a malicious executable may be carefully constructed so that an ML model misclassifies it as benign, despite its truly malicious nature. Such an attack has important ramifications for the safety of machine learning systems, especially considering the well-established adversarial nature of malware detection. Adversarial attacks are not exclusive to ML-based malware detection; they apply across all domains. Thus, significant effort has been devoted to developing methods for defending ML models against adversarial attacks. However, as we discuss, these have a number of limitations, such as their ineffectiveness, failure to cope with newer types of attacks, and reduced performance on legitimate queries. Furthermore, in ML-based malware detection, defenses against adversarial attacks have received substantially limited attention, despite the threat posed by these attacks.

In this dissertation, we aim to establish a fresh understanding of methods to defend against adversarial attacks in ML-based malware detection. We propose two novel defensive methods based on promising approaches, moving target defenses (MTDs) and stateful defenses, which are based on other areas of cybersecurity and have never been applied to this domain before. Particularly, we first propose a novel strategic moving target defense against adversarial attacks that overcomes the challenges and limitations of related work. We show that this strategic defense outperforms several prior defenses in the first evaluation of its kind, which considers a range of threat models. Second, we characterize and evaluate, for the first time, the effectiveness of several MTDs for adversarial ML applied to this domain. In a comprehensive study, we extensively compare their performances with each other and with other types of defenses for adversarial ML. We expose the vulnerability of these MTDs to both existing attack strategies and our proposed novel ones, highlighting the key weaknesses of these approaches. Based on these findings, we present key recommendations for advancing work on MTDs against adversarial ML. Inspired by one of these recommendations, we then propose a novel stateful defense against adversarial attacks on ML-based malware detection. In the first study of its kind, we showcase our defense's capabilities under several attack scenarios, demonstrating its ability to reduce attack success in a manner that is superior to a range of prior stateful defenses.

This dissertation, when taken as a whole, offers a fresh perspective on the various methods that may be considered promising directions for coping with adversarial attacks on ML-based malware detection.

CONTENTS

List of Figures	x
List of Tables	xii
List of Abbreviations	xiii
1 Introduction	1
1.1 Overview	1
1.2 Motivation	2
1.3 Research Aims	3
1.4 Contributions	5
1.5 Publications	6
1.6 Outline of Dissertation	7
2 Background & Literature Review	9
2.1 Machine Learning	9
2.1.1 Machine Learning Tasks	9
2.1.2 ML-based Malware Detection vs. Other Malware Detection Methods	10
2.1.3 Anatomy of ML-based Malware Detection	11
2.1.4 Evaluating ML Classifiers	12
2.1.5 Further Challenges & Considerations	13
2.2 Attacking ML Models	14
2.2.1 Threat Models	14
2.2.2 Evasion Attacks Against ML Models	16
2.2.3 Evasion Attacks Against ML-based Malware Detection	17
2.2.4 Transferability of Adversarial Examples	19
2.2.5 Attack Strategies Under Attacker Scenarios	20
2.2.6 Evaluating Attack Performance	23
2.3 Defending ML Models	24
2.3.1 Single-Model Defenses (Model Robustness)	25
2.3.2 Ensemble Defenses	26

2.3.3	Attack Detection & Stateful Defenses	28
2.4	Focus of this Dissertation & Related Work	29
2.4.1	Strategic Defense Against Adversarial Attacks	30
2.4.2	Effectiveness of Moving Target Defenses Against Adversarial Attacks	33
2.4.3	Stateful Defense Against Adversarial Query Attacks	35
2.5	Conclusion	37
3	Strategic Defense Against Adversarial Attacks	39
3.1	Introduction	39
3.2	Threat Model	40
3.2.1	Application Model	41
3.2.2	Attacker's Goal	41
3.2.3	Attacker Capabilities & Knowledge	41
3.3	StratDef	42
3.3.1	What to Move: Phase 1 — Model Generation	42
3.3.2	What to Move: Phase 2 — Model Selection	44
3.3.3	How to Move: Devising a Strategy	45
3.3.4	When to Move: Making a Prediction	48
3.4	Experimental Setup	48
3.4.1	Datasets	48
3.4.2	Training Models & Defenses	49
3.4.3	Generating Adversarial Examples	51
3.4.4	Modeling Gray-box Attacker Profiles	53
3.5	Evaluation & Results	54
3.5.1	Performance of StratDef	54
3.5.2	StratDef vs. Other Defenses	56
3.5.3	Efficiency of StratDef	58
3.5.4	StratDef vs. UAPs	59
3.5.5	StratDef vs. Black-box Attacks	60
3.6	Conclusion	62
4	Effectiveness of Moving Target Defenses Against Adversarial Attacks	63
4.1	Introduction	63
4.2	Threat Model	64
4.2.1	Target Model	65
4.2.2	Attacker's Goal	65
4.2.3	Attacker Capabilities & Knowledge	65
4.3	Attack Strategies	66
4.3.1	Transferability Attacks	66

4.3.2	Query Attacks	69
4.4	Experimental Setup	71
4.4.1	Datasets	71
4.4.2	Moving Target Defenses	72
4.4.3	Performance of MTDs in Non-Adversarial Conditions	73
4.4.4	Other Evaluated Defenses	73
4.4.5	Substitute Models for Transferability Attack	74
4.4.6	Generating Adversarial Examples	74
4.4.7	Evaluation Metrics	75
4.5	Black-box Results	76
4.5.1	Transferability Attack	76
4.5.2	Query Attack	79
4.6	Gray-box Results	81
4.6.1	Transferability Attack	81
4.6.2	Query Attack	82
4.6.3	Universal Adversarial Perturbations	83
4.7	Beyond Adversarial Robustness	84
4.7.1	Procedure	84
4.7.2	Results	84
4.8	Fingerprinting & Reconnaissance	87
4.8.1	Determining Predictive Nature of Defenses	87
4.8.2	Determining Movement of Hybrid Defenses	88
4.9	Discussion & Recommendations	88
4.10	Conclusion	90
5	Stateful Defense Against Adversarial Query Attacks	92
5.1	Introduction	92
5.2	Threat Model	93
5.2.1	Target Model	93
5.2.2	Attacker's Goal	94
5.2.3	Attacker Capabilities & Knowledge	94
5.3	MalProtect	94
5.3.1	Query History	95
5.3.2	Indicators for Analyzing Query History	96
5.3.3	Attack Detection	98
5.3.4	Defensive Action & Prediction	99
5.4	Experimental Setup & Preliminaries	99
5.4.1	Datasets	99
5.4.2	MalProtect Configurations	100

5.4.3	Other Stateful Defenses	101
5.4.4	Prediction Models	102
5.4.5	Performing Query Attacks	103
5.4.6	Performing Other Attacks	103
5.5	Black-box Query Attack Results	104
5.6	Gray-box Query Attack Results	106
5.7	Evaluation of Additional Query Attacks	108
5.8	Interpretability & Adaptive Attack	109
5.8.1	Interpretability of MalProtect	109
5.8.2	Attack Rationale	110
5.8.3	Adaptive Attack Strategy	110
5.8.4	Results	111
5.9	Beyond Adversarial Robustness	111
5.9.1	Procedure	111
5.9.2	Results	112
5.10	Overheads of MalProtect	114
5.10.1	Storage	114
5.10.2	Prediction Time	114
5.10.3	Optimization	114
5.11	Conclusion	114
6	Conclusions & Future Work	116
6.1	Conclusions	116
6.2	Future Work	118
	References	120
	Appendix A Permitted Perturbations for Android Datasets	137
A.1	Permitted Perturbations for Android Datasets	137
	Appendix B Strategic Defense Against Adversarial Attacks	138
B.1	Examples of StratDef Strategies	138
B.2	Extended Results	139
	Appendix C Effectiveness of Moving Target Defenses Against Adversarial Attacks	140
C.1	Effectiveness of Attacks Against Models with Fewer Features	140
C.2	Extended Results	140
C.3	Varying Training Data Size in Gray-box Transferability Attack	140
	Appendix D Stateful Defense Against Adversarial Query Attacks	142
D.1	Query Attack Strategies	142

LIST OF FIGURES

2.1	Overview of ML-based Malware Detection Classifier	11
2.2	Overview of Evasion Attacks	17
2.3	Evasion Attack Pipeline in Malware Detection Domain	18
2.4	Transferability Attack Pipeline in Malware Detection Domain	21
2.5	Query Attack Pipeline in Malware Detection Domain	22
2.6	Overview of a Voting Defense	27
2.7	Overview of a Moving Target Defense	27
2.8	Overview of a Stateful Defense	29
3.1	Overview of StratDef	43
3.2	Average FPR of Voting Strategies Against Strong Attacker.	50
3.3	Accuracy of StratDef Against Different Attackers	55
3.4	Average Metric Values for StratDef Against Strong Attacker	56
3.5	Comparison Between StratDef & Other Defenses Against Strong Attacker	57
3.6	Average Prediction Time Against Strong Attacker	58
3.7	Average Memory Consumption Against Strong Attacker	59
3.8	Accuracy of StratDef & Other Defenses Against Universal Attacker	60
3.9	Black-box Attack Against StratDef & Other Defenses	61
4.1	Overview of Our Transferability Attack Strategy	67
4.2	Overview of Our Gray-box Query Attack Strategy	70
4.3	Malware Detection Performance of MTDs	73
4.4	Evasion Rate of Black-box Transferability Attack Strategies	77
4.5	Evasion Rate of Black-box Transferability Attack Strategy (Variable)	78
4.6	Average RER of Black-box Transferability Attack	79
4.7	Evasion Rate of Black-box Query Attack Strategy	80
4.8	Average Queries for Evasion (Black-box) vs. Maximum Queries Permitted	80
4.9	Evasion Rate of Gray-box Transferability Attack Strategies	81
4.10	Evasion Rate of Gray-box Query Attack Strategy	82
4.11	Average Queries for Evasion (Gray-box) vs. Maximum Queries Permitted	83
4.12	Evasion Rate of Attack (UAPs)	84

4.13	Accuracy, FPR, F1 & AUC of MTDs	86
4.14	Fingerprinting Example	88
5.1	Overview of MalProtect	95
5.2	Black-box Query Attack Results	105
5.3	Gray-box Query Attack Results	107
5.4	Evasion Rate of Additional Query Attacks	109
5.5	Influence of MalProtect Indicators	109
5.6	Evasion Rate of Adaptive Attack	111
5.7	Accuracy, FPR, F1 & AUC of Stateful Defenses	113

LIST OF TABLES

2.1	Overview of Related Literature for MTDs	31
2.2	Overview of Related Literature for Stateful Defenses	36
3.1	Example of Global Strategy Vector	46
3.2	Architectures of Vanilla Models	49
3.3	Attacks Used to Generate Adversarial Examples	52
3.4	Profiles of Different Attackers	53
4.1	Configuration of MTDs	72
4.2	Configuration of Other Evaluated Defenses	74
4.3	Architectures of Vanilla & Substitute Models	74
5.1	Decision Models for MalProtect Configurations	101
5.2	Thresholds for Detection for SD	102
5.3	Architectures of Prediction Models	102
5.4	Architectures of Vanilla Models	104

LIST OF ABBREVIATIONS

API	Application Programming Interface
AT	Adversarial Training / Adversarially-trained
AUC	Area Under (ROC) Curve
DNN	Deep Neural Network
DT	Decision Tree
FNR	False Negative Rate
FPR	False Positive Rate
ML	Machine Learning
MTD	Moving Target Defense
NLP	Natural Language Processing
NN	Neural Network
OBJ	Research Objective
OS	Operating System
RF	Random Forest
RFN	Random Feature Nullification
ROC	Receiver Operating Characteristic
SVM	Support Vector Machine
TNR	True Negative Rate
TPR	True Positive Rate
URL	Uniform Resource Locator

CHAPTER 1

INTRODUCTION

In recent years, machine learning (ML) has yielded enormous capabilities across a number of fields [95]. These include, but are not limited to, computer vision [114, 96], natural language processing [154], finance and economics [98], healthcare [83, 218], and others. Whether it is recognizing speech to activate a smart personal assistant, removing unwanted objects from an image, or maintaining a vehicle’s lane on the road, the various applications of ML are becoming increasingly embedded in our daily lives. Since organizations and businesses now have access to significantly more computational resources, larger swaths of data, and cutting-edge algorithmic solutions, ML has made it possible for them to take advantage of the significant advancements that it offers in order to improve their systems and processes. Because of this, fundamental aspects of business, such as automation and scalability, have been redefined [95]. Much of this evolution has been fueled by cloud-based ML, especially with Machine-Learning-as-a-Service (MLaaS) platforms that have allowed developers to comfortably leverage the power of remotely-hosted models with minimal costs [233].

Additionally, disciplines such as cybersecurity have undergone a revolution in their approach to tackling challenging problems. ML-based methods are being readily used by researchers and security practitioners to improve the security of systems and foster innovation in this domain. There is a plethora of problems in cybersecurity that deal with separating *malicious* objects from those that are *benign* [16], such as detecting spam emails [63], locating malicious and abusive social media content [149], maintaining intrusion detection systems and web application firewalls [77, 141], and *detecting malware* [153, 88]. Specifically, for malware detection — which categorizes software executables as *benign* or *malicious* — ML offers several enhancements over other detection methods [173, 236, 145, 151, 60], such as improved classification and detection performance, the proven ability to assimilate large datasets rapidly, and the capacity to generalize to previously unknown or unseen threats, thereby allowing for greater explanations of patterns of behavior. Moreover, resource-consuming processes such as the reverse engineering of software executables can be automated with ML-based approaches, freeing up time that can be better spent understanding the threat environment [225]. Accordingly, several solutions have been proposed for ML-based malware detection that have been applied across various platforms, such as Android [153, 18, 10], Windows [32, 8, 172], PDFs [47, 240, 189], and more. Not only is ML-based malware detection being actively worked on in academic research, but security practitioners in industry are actively harnessing the capabilities of ML to prevent malicious applications from spreading across their services and platforms (e.g., [211, 1]).

However, despite the vast benefits and great potential of machine learning, recent work has shown that its use can *considerably* widen the attack surface at all stages of the learning pipeline through *adversarial ML attacks* [162, 207, 28, 74, 77, 76]. There are two main categories of adversarial attacks: with *poisoning attacks* (sometimes referred to as *backdoor attacks*), an adversary manipulates a model’s behavior by conducting an attack at training-time that influences and meddles with future predictions [162]. Meanwhile, with *evasion attacks*, an attacker deceives and *fools* a classifier at test-time [162]. For example, the image recognition model of a self-driving car may be deceived into misclassifying a stop sign as a speed limit sign [77], or a speech recognition model may be tricked into allowing adversaries to perform actions that are reserved for privileged users [187].

In cybersecurity, the operating environment is adversarial *by nature*, where attackers are *always* present and looking for new attack vectors. As the usage of machine learning models in malware detection systems increases, attackers have additional avenues to conduct sophisticated attacks. They may try to evade ML-based detection systems by developing malicious executables in a particular manner such that the malicious functionality of an executable is preserved while an ML model classifies that malicious executable as benign [207, 88, 32]. Such an executable or sample is known as an *adversarial example*. Thus, an attacker can evade ML classifiers and cause them to predict an input sample as *benign* when, in fact, it is *malicious* [88]. In addition, attackers have the ability to devote a large amount of effort to reconnaissance work to model the attack surface and methodically organize their attacks. As a consequence, adversarial machine learning poses a risk that is both realistic and significant to the security of ML-based systems.

1.2 Motivation

It is essential to maintain a high degree of resilience against adversarial ML attacks for effectively applying ML to cybersecurity. To this end, several defensive approaches have been proposed that attempt to increase the robustness of ML models against adversarial attacks across various domains. However, as we discuss in detail later in Chapter 2, a number of limitations of these defensive methods have been exposed. These include the ineffectiveness of defenses when they are applied in different contexts (e.g., different base models, domains, datasets, etc.), the worsening of the performance of robust models on legitimate data, the inability to detect attacks in progress, and the susceptibility of models to pertinent threats such as the transferability of adversarial examples. Moreover, most defensive approaches have aimed to increase the robustness of individual models with static and non-strategic methods that fail to cope with the latest threats, which means that in this constant cat-and-mouse game, they are broken by newer types of adversarial attacks [21, 97, 46]. Even more concerning is the fact that, despite the well-established threat landscape, defenses for ML-based malware detection systems have received considerably less attention. In fact, less than one percent of the papers on adversarial ML in the last eight years relate to ML-based malware detection [42].

Hence, it is abundantly evident that machine learning models, particularly for ML-based malware detection, require better protection against adversarial attacks. For this purpose, in this dissertation, we draw inspiration from other areas of cybersecurity, such as *moving target defenses (MTDs)* that continually vary their configuration and use different techniques to behave non-statically. As we discuss later, MTDs are based on the principle that a system is easier to defend if the attack surface is constantly changing, since attackers would then

have it harder to find vulnerabilities and profile the system [150, 58]. For adversarial ML, defensive approaches such as MTDs can regularly *move* their configuration, such as how predictions are produced or the ML models used to make them, in order to ensure that they do not remain static, fixed targets. This increases the complexity for attackers to conduct targeted attacks, steal models, and perform other reconnaissance activities. Therefore, in this dissertation, we propose a strategic moving target defense for adversarial ML, which aims to reduce the evasion of evasion attacks in ML-based malware detection.

While MTDs may make it more difficult for attackers to discover vulnerabilities and conduct attacks, additional methods are required to detect adversarial behavior as it happens online. For this purpose, *stateful* detection methods can be used that maintain a record of different aspects of a system to detect malicious activity. In the context of adversarial ML, this can be achieved by storing the sequence of queries received by the system using *state* and analyzing them to prevent the generation of adversarial examples. Hence, to detect attacks in progress, we draw inspiration from stateful firewalls and network intrusion detection systems, which operate by detecting harmful inputs before they can reach an underlying system. This is achieved by analyzing the system inputs using different techniques to discover malicious activity. Therefore, to detect adversarial ML attacks in progress, we propose a novel *stateful* defense for the ML-based malware detection domain that analyzes the user queries to detect anomalous and adversarial behavior. Intriguingly, most proposed defenses are *stateless* (and do not have access to prior user queries), with their only aim being to increase model robustness against evasion attacks rather than detect *attacks in progress*.

Together, our proposed defensive approaches aim to increase the robustness of ML-based detection systems against adversarial attacks while also trying to detect and prevent them in the first place.

1.3 Research Aims

In this dissertation, our objective is to enhance the effectiveness of defending ML-based malware detection classifiers against adversarial attacks by proposing novel defenses and developing a deeper understanding of different methods for this problem.

We propose two novel defensive methods based on promising approaches with the aim to improve robustness against adversarial attacks on ML-based malware detection: one approach based on *moving target defenses (MTDs)* and another on *stateful defenses*, neither of which have been applied to the ML-based malware detection domain before. These approaches are based on other areas of cybersecurity and have demonstrated potential to defend against adversarial ML attacks in other domains (although prior proposed solutions based on these approaches have their own limitations, as we discuss later). MTDs (e.g., [174, 191, 13, 200]) seek to increase the uncertainty and complexity for attackers, making reconnaissance and targeted attacks less successful by using multiple constituent ML models in a particular manner to produce predictions [150, 58]. Meanwhile, stateful defenses (e.g., [52, 130, 110]) analyze queries for abnormalities using different statistical and ML-based methods to protect ML prediction models and detect adversarial attacks, which have been heralded as a promising way forward to deal with adversarial behavior [37]. Therefore, in this dissertation, we propose novel defenses based on these approaches.

Furthermore, in order to provide context for the performance of our novel defensive approaches, it is imperative to assess how they compare to prior defenses, taking into consideration the malware detection domain's particular characteristics and constraints regarding the generation of adversarial examples. Hence, as part of our work, we conduct key evaluations, which are the first of their kind, comparing and characterizing prior related work that proposes defensive methods based on MTDs and stateful defenses, as well as other defensive approaches that do not use these techniques. For the first time, we aim to study several MTDs proposed for adversarial ML as a whole, thereby deriving key insights about their effectiveness against attacks on ML-based malware detection, in order to understand how to develop the best-performing MTDs. Moreover, we aim to develop a thorough understanding of how different stateful defenses operate in this domain by comparing and characterizing the performance of a number of proposed defenses. By studying the strengths and limitations of various defenses, we aim to better understand how the security and robustness of ML models can be improved in light of the emerging threats posed by adversarial ML.

In summary, we propose two novel defensive approaches, based on moving target defenses and stateful defenses. We propose a strategic MTD, where we investigate various factors, such as what, how, and when it should adapt itself, with the goal of making the attacker's job more difficult by increasing the uncertainty and complexity of the problem. We provide an entire framework that considers the non-trivial processes related to selecting the constituent models of the ensemble and then how to strategically use them, which have been overlooked in prior work. Moreover, we propose a stateful defense that goes above and beyond prior detection techniques by employing several threat indicators that conduct an analysis to detect if there is an attack in progress. We aim to increase the detection of attacks reliably, while addressing key limitations of prior work, such as insufficient detection mechanisms, poor maintenance of the query history, and weaknesses regarding the use of account-based detection, among others. We investigate whether these proposed defenses can effectively defend against adversarial attacks in the ML-based malware detection domain in comprehensive studies, which are the first of their kind, comparing several related defensive approaches. We outline our research goals and objectives as follows¹:

- **Objective 1 (OBJ1) — Development of a Strategic Moving Target Defense:** Can a novel strategic moving target defense be designed to increase the robustness and defensive capability against adversarial attacks in ML-based malware detection? Can key aspects of MTDs such as movement mechanisms and strategies, model heterogeneity, and redundancy be considered in the design in order to minimize the transferability of adversarial examples across constituent models? Is it possible to tailor such a defense to different threat levels?

We accomplish this in Chapter 3.

- **Objective 2 (OBJ2) — Evaluation of Proposed Strategic Moving Target Defense for Adversarial ML:** How effective is our proposed strategic MTD compared with a range of prior defenses? That is, how does it perform under different threat models when the constraints and characteristics of the domain are properly considered? Prior evaluations and comparisons of defenses either do not consider

¹These are not listed in order of importance but in their approximate chronological order within the dissertation.

the constraints of the domain sufficiently or do not compare a range of different defense approaches in a comprehensive manner, which also makes it difficult to draw comparisons between them.

We accomplish this in Chapter 3.

- **Objective 3 (OBJ3) — Evaluation & Characterization of MTDs for Adversarial ML:** How do different MTD approaches compare with each other and other types of defenses, and is there any scope to characterize this performance? How does the performance of MTDs vary under different attack scenarios? Are there any key recommendations that can be derived from an empirical study of MTDs that lead to more effective design and construction of MTDs?

We accomplish this in Chapter 4.

- **Objective 4 (OBJ4) — Development of a Stateful Defense:** Can a novel stateful defense against adversarial ML attacks be designed to defend effectively in the ML-based malware detection domain? Does a stateful defense need to be adapted or have its detection approach modified to operate effectively in this domain? What are the challenges associated with developing such a defense in this domain?

We accomplish this in Chapter 5.

- **Objective 5 (OBJ5) — Evaluation of Proposed Stateful Defense for Adversarial ML:** How effective is our proposed stateful defense at thwarting adversarial attacks in ML-based malware detection? How does this compare with prior stateful detection approaches under different attacker scenarios? Can the performance of stateful defenses as a whole be characterized?

We accomplish this in Chapter 5.

1.4 Contributions

The primary contributions of this dissertation to the current state of research are presented as follows:

- We propose the first strategic defense against adversarial attacks in the malware detection domain. Our defense, StratDef, is based on an MTD approach where we propose different strategic and heuristically-driven methods for determining what, how, and when a defense system should move to achieve a high degree of adversarial robustness. This includes key steps related to model selection and the development of strategies. This contribution relates to OBJ1.

We present work on this in Chapter 3.

- We offer a detailed evaluation of prior defensive approaches to demonstrate the necessity of a strategized approach. That is, we consider the constraints and characteristics of this domain in a proper manner, unlike prior evaluations, and evaluate a range of defenses. The results show that our strategized defense can increase accuracy by 50+% in the most adverse conditions for both Android and Windows malware. This contribution relates to OBJ2.

We present work on this in Chapter 3.

- We are the first to evaluate how a strategized MTD fares against a variety of attackers, such as gray-box attackers with limited knowledge, black-box attackers with zero-knowledge, and attackers who only use adversarial examples generated with Universal Adversarial Perturbations (UAPs). This contribution relates to OBJ1 and OBJ2.

We present work on this in Chapter 3.

- We conduct the first systematic evaluation of several MTDs to defend against adversarial ML attacks applied to the malware detection domain. This is achieved by using existing transferability and query attack strategies as well as novel improvements to these strategies for maximizing the evasion of MTDs. Across Android and Windows malware and goodware, we show that the MTDs can be evaded with minimal information about them. This contribution relates to OBJ3.

We present work on this in Chapter 4.

- We show that it may be possible to fingerprint and recognize MTDs with a set of initial techniques that allow us to discover the predictive nature of the MTDs studied and, in some cases, some of their critical hyperparameters. This contribution relates to OBJ3.

We present work on this in Chapter 4.

- Informed by the evidence produced in our systematic evaluation of MTDs, we derive key insights and make crucial recommendations to help design more effective MTDs against adversarial ML attacks in the malware detection domain. This contribution relates to OBJ3.

We present work on this in Chapter 4.

- We propose the first stateful defense against adversarial query attacks in the malware detection domain. Our defense, MalProtect, uses several threat indicators to reliably detect query attacks and prevent the generation of adversarial examples. This contribution relates to OBJ4.

We present work on this in Chapter 5.

- We provide the first evaluation of several existing stateful defenses applied to the ML-based malware detection domain. The results show that MalProtect significantly reduces the evasion rate of query attacks across Android and Windows under a range of threat models and attacker scenarios. This contribution relates to OBJ5.

We present work on this in Chapter 5.

1.5 Publications

The following research papers are under review at conferences and journals as a result of work towards this dissertation:

- Rashid, Aqib; Such, Jose: “*Effectiveness of Moving Target Defenses for Adversarial Attacks in ML-based Malware Detection*”. Submitted, under review.

The following research papers have been published/accepted at conferences and journals as a result of work towards this dissertation:

- Rashid, Aqib; Such, Jose: “*MalProtect: Stateful Defense Against Adversarial Query Attacks in ML-based Malware Detection*”. Accepted in IEEE Transactions on Information Forensics and Security, 2023.
- Rashid, Aqib; Such, Jose: “*StratDef: Strategic Defense Against Adversarial Attacks in ML-based Malware Detection*”. Accepted in Computers & Security, 2023.

1.6 Outline of Dissertation

The outline of this dissertation is as follows:

- In Chapter 2, we present the background information and literature review needed to understand and appreciate the work and contributions in the remainder of the dissertation. We introduce the core concepts related to machine learning before delving into machine learning applications in cybersecurity and, more specifically, ML-based malware detection. We then analyze the problem of adversarial machine learning and investigate how attacks against ML models are actually performed. Additionally, we review the defensive approaches that have been previously proposed for coping with this problem. We also provide a comprehensive literature review of related work to ours.
- In Chapter 3, we propose StratDef, which is a novel strategic defense system for the malware detection domain based on a moving target defense approach. We overcome the challenges related to the systematic construction, selection, and strategic use of models to maximize adversarial robustness. We provide the first comprehensive evaluation of defenses against adversarial attacks on machine learning for malware detection, where our threat model explores different levels of threat, attacker knowledge, capabilities, and attack intensities. We show that StratDef performs better than other defenses even when facing the peak adversarial threat. We also show that, of the existing defenses, only a few adversarially-trained models provide substantially better protection than just using vanilla models but are still outperformed by StratDef.
- In Chapter 4, for the first time, we study and compare the effectiveness of several recent MTDs against adversarial ML attacks. Under different threat models, we show that transferability and query attack strategies are still able to achieve high levels of evasion against these defenses. This is achieved by using a combination of prior attack strategies and novel ones that we propose in order to increase the evasion of models. We also show that fingerprinting and reconnaissance of MTDs is possible and demonstrate how attackers may obtain critical defense hyperparameters as well as information about how predictions are produced. Based on our findings, we present key recommendations for spurring future work on the development of effective MTDs for adversarial attacks in ML-based malware detection.
- In Chapter 5, we present MalProtect, which is a novel stateful defense for the malware detection domain. We show that MalProtect can significantly reduce the evasion rate of query attacks under a range of

attacker scenarios and threat models. To do this, MalProtect uses several threat indicators to detect attacks and prevent the generation of adversarial examples. In a comprehensive evaluation, we show that MalProtect outperforms a range of prior stateful defenses, especially under the peak adversarial threat.

- Finally, in Chapter 6, we conclude the dissertation by providing a summary of the work. We also discuss directions for future research.

CHAPTER 2

BACKGROUND & LITERATURE REVIEW

Our work focuses on exploring defenses against adversarial attacks on ML-based malware detection. In this chapter, we cover the essential concepts, methods, and work associated with this topic and lay the framework for the reader to better understand the work presented later in this dissertation. To accomplish this, Section 2.1 provides the background related to the application of machine learning algorithms in a security context. There, we discuss ML-based malware detection, its broad application, and the advantages it offers over other malware detection methods, such as rule-based and signature-based malware detection. Then, in Sections 2.2 and 2.3, we extensively discuss adversarial machine learning and related work. We explore how, due to vulnerabilities in ML algorithms, classifiers can be exploited and *evaded*. Adversarial machine learning has increased the cybersecurity arms race between defenders trying to protect their models from adversarial behavior, and attackers who try to evade detection. We present important concepts from the literature, such as threat models, attacker capabilities, attacker knowledge, and evaluation metrics. Following this, in Section 2.4, we present a literature review of the research work that is most related to each of our key contributions.

2.1 Machine Learning

We introduce the essential concepts and terminology related to machine learning and its application to the cybersecurity domain, and more specifically, to malware detection.

2.1.1 Machine Learning Tasks

Machine learning (ML), in the broadest sense, is the method by which an algorithm solves a particular task using some dataset without being given direct instructions on how to do so. In numerous fields, it has achieved exceptional success [95, 57, 162]. ML has also played an increasing role in cybersecurity; for example, in finding spam tweets [149], threat hunting [49], network intrusion detection [217], and more. Machine learning has also been widely applied to *malware detection* — which is the focus of our work. With this, an ML model can predict whether a particular file or executable is *benign* or *malicious* [82].

Typically, machine learning tasks are either *supervised* or *unsupervised* [33]. In a supervised learning setting, the ML algorithm is provided with a set of *input samples* and their corresponding *labels* that, when put together, form the *training data* [114, 206]. The ML algorithm learns the association between input samples and labels from the training data in order to infer labels for unseen input samples in the future. This is achieved by

recognizing the patterns and correlations in the individual properties and characteristics of data, known as *features*. Supervised learning tasks include *classification*, which produces categorical predictions (e.g., whether a file is benign or malicious), while *regression* algorithms produce numerical predictions (e.g., the likelihood of an attack in progress). In contrast, in an unsupervised learning setting, an algorithm can cluster input samples according to some notion of similarity [107], such as the analysis of the malware family of an input sample [78]. Furthermore, *semi-supervised learning* has been proposed, where a small amount of labeled data is combined with a large amount of unlabeled data [237]. This is useful in instances where acquiring the class labels is challenging and requires specialized knowledge.

In this dissertation, we focus on supervised learning — and more specifically, classification — as we are interested in the problem of distinguishing benign objects from malicious ones in the context of ML-based malware detection. Under this learning setting, an ML model is constructed using the training data, which consists of input samples and labels. The ML model can then predict whether an unseen input sample belongs to the benign or malware class through the patterns and correlations it has learned during its training.

2.1.2 ML-based Malware Detection vs. Other Malware Detection Methods

Besides machine learning, rule-based and signature-based detection are techniques that have been frequently employed to detect malware [236, 145, 151, 60]. These detection methods use databases of regular expressions, patterns, and other signatures from the data and code of executables that are known to be malicious to identify fingerprints and sequences (e.g., bytes or instructions) in other samples. An executable is then deemed malicious if a signature stored in the database can be identified from the executable's code [7] or if it matches other criteria relating to known threats [173].

However, ML-based detection can surpass the performance of these detection methods for a variety of reasons. Firstly, rule-based and signature-based detection systems depend on databases that must be regularly updated with the most recent threats. This renders them less adaptable and capable of recognizing previously undiscovered threats [84, 173, 4, 20]. Conversely, a machine learning model may recognize patterns that a human analyst would be unable to recognize, while consistent re-training of the model may allow it to keep up with the latest threats, which might lead to an increased detection rate by ML-based solutions [15]. Moreover, ML-based malware detection is versatile, as it can be more easily applied to a range of applications and technologies. This is unlike traditional signature-based approaches [20], which may require separate databases/stores of data for identifying malicious objects of different categories. Additionally, ML models are able to offer fewer false positives because they do not look for specific patterns of code or data. This is especially important in this domain [90, 202, 229, 54, 125] as it leads to a higher quality of service with users less frequently finding their benign files and executables misclassified as malicious.

Thus, ML-based approaches offer several advantages over other detection methods, such as the ability to assimilate large datasets rapidly and to generalize to unknown threats that may have remained undetected previously [181]. These advantages make ML-based malware detection significantly more capable and desirable to employ. We next describe in detail how ML-based malware detection operates.

However, ML-based malware detection is not a silver bullet for detecting malicious software and executables. Later, in Section 2.1.5, we discuss some of the related challenges and considerations.

2.1.3 Anatomy of ML-based Malware Detection

With a supervised learning approach, an ML model can be trained to predict whether an input sample belongs to the *benign* class (also referred to as *goodware*), or the *malware* class. ML-based malware detection operates in the feature-space as it uses mathematical representations of software executables, through *features*, to make predictions for unseen samples. For a visual explanation of this procedure, see Figure 2.1, where an overview of the procedure for constructing an ML-based malware detection classifier is presented.

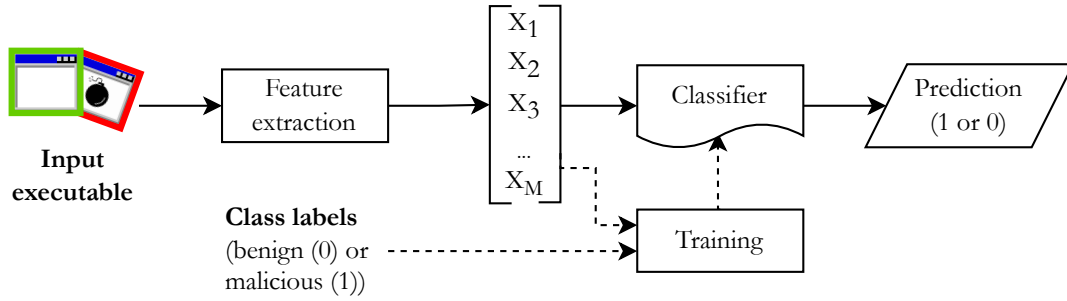


Figure 2.1: Overview of an ML-based malware detection classifier.

This figure shows that *feature extraction* is utilized to develop representations of software executables and samples, which can be used to train ML classification models. A number of other *preprocessing* steps may also be applied, but the overall objective is to transform a software executable from the *problem-space* (i.e., the actual executable) into a binary *feature vector* for the *feature-space* where machine learning algorithms operate [207, 168]. The feature vector representing a software executable consists of binary (or *discrete*) feature values corresponding to its various characteristics¹. For example, the usage and presence of APIs, libraries, system calls, resources, or the accessing of network addresses, as well as the actual code itself, are typically parsed as features. In formal terms, therefore, a feature vector X for a software executable is such that $X \in \{0, 1\}^M$, where $X_i = 1$ indicates the presence of feature i and $X_i = 0$ indicates its absence. A number of different samples (i.e., the feature vectors) and their associated class labels (benign or malware) from a dataset can then be used to train an ML model. As malware detection is a binary classification problem, the classes (i.e., labels) to be considered are the *positive* class (1 for *malware*) and the *negative* class (0 for *benign*). All of this data is typically aggregated into a *dataset* (e.g., [11, 18, 8]), which provides information about thousands of software samples from a particular platform. The quality of the trained classifier and the predictions it produces hinge precisely on the features used to construct the model [88, 8, 181, 69].

¹This is as opposed to *continuous* feature values in other domains, e.g., image recognition. In that domain, the pixel values of the image are used as features.

Typically, the dataset is partitioned using some ratio into training, validation, and test sets², such as the Pareto principle. With the training set, ML classifiers and models such as deep neural networks (DNNs), support vector machines (SVMs), decision trees, random forests, and others can be trained. These ML models and classifiers can be constructed using a range of different settings, configurations, and *hyperparameters* [33]. Examples of hyperparameters include the learning rate and batch size in DNNs, regularization terms in SVMs, or the quantity of trees in a random forest. Generally, though, a model requires several phases of tuning by evaluating its *loss function* in order to reach a desirable level of performance for deployment. The loss function is a mathematical function that is used to quantify the difference (i.e., *error*) between the expected output of a model and the actual output. When training a machine learning model, the goal is to minimize the loss function's value [33]. As part of this, several hyperparameters and decision thresholds are examined using various performance metrics for achieving the desired performance.

2.1.4 Evaluating ML Classifiers

It is necessary to perform assessments of the quality of the predictions that an ML model produces. For this purpose, a *validation set* and a *test set* containing input samples from the same distribution as the training data are utilized. During the development phase, the model's performance is assessed by having it produce predictions for samples from the validation set. If the model's performance falls below the intended or anticipated level, the hyperparameters and configuration are adjusted, and the model is re-evaluated on the validation set as part of an ongoing cycle of development and improvement. Therefore, the model is tuned based on its performance on this data. Once the model's performance on the validation set reaches the desired level, a final assessment of it is conducted using the completely unseen test set prior to its deployment. This approach is intended to limit *overfitting*, which occurs when a model is overtuned on local data, preventing it from generalizing to unseen samples. This means that it performs well under development but inadequately against unseen samples in the wild. With a completely independent test set, the model's performance can be assessed on unseen samples that were not used to tailor its performance during its construction, therefore avoiding overfitting.

To facilitate the examination and comparison of models, metrics that are easier to interpret are typically employed. A core metric is *accuracy*, which measures the proportion of correct predictions. In the malware detection domain, this is a measure of the proportion of predictions that match the true label (e.g., benign or malware). However, it is not appropriate to rely only on a single metric, as the binary classification problem of malware detection is multifaceted [175]. As the possible predictions in binary classification tasks are either *positive* or *negative*, the classifier's predictions can only be correct (therefore, true positives (TP) and true negatives (TN)), or erroneous (therefore, false positives (FP) and false negatives (FN)). These metrics allow for the development of the *true positive rate* (i.e., the proportion of samples that were correctly predicted as positive) and the *false positive rate* (i.e., the proportion of samples that were incorrectly predicted as positive). Within the malware detection domain in particular, FPR must remain low [90, 202, 229, 125] lest a system be deployed that incorrectly (and frustratingly) flags legitimate queries and input samples as malicious, thereby affecting the quality of service. The *receiver operating characteristic (ROC) curve*, which shows the TPR over

²In malware detection, there are also different ways to split the dataset into training and test data (see Section 2.1.5 later).

the FPR, demonstrates the intrinsic trade-off between these two measurements. Classifiers that have a greater area under the ROC curve (known as the *AUC*) will demonstrate superior performance at distinguishing between the classes [94].

Alternatively, *precision* and *recall* can be employed, with precision capturing the proportion of correct positive predictions and recall capturing the proportion of positive examples captured. Recall intuitively reflects the probability that no positive examples were ignored, whereas precision evaluates the accuracy of positive predictions. The formulas for these metrics are as follows:

$$Precision = \frac{TP}{TP + FP} \quad (2.1)$$

$$Recall = \frac{TP}{TP + FN} \quad (2.2)$$

Similar to TPR and FPR, the *F1-score* captures the trade-off between precision and recall. The F1-score represents the harmonic mean of two scores. It is calculated as follows:

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN} \quad (2.3)$$

Thus far, we have discussed how ML-based malware detection classifiers are constructed and evaluated. However, there exist several unique challenges that require consideration, which we discuss next.

2.1.5 Further Challenges & Considerations

In many security applications, sampling from the true distribution (or real-world data) is extremely difficult and sometimes even impossible [19, 20]. In particular, the lack of publicly-available, up-to-date datasets in the malware detection domain is a well-known general problem, which limits the merits and conclusions of academic work in this domain [19, 20]. Therefore, academic work in this domain often utilizes datasets that may not represent the latest threats. To partially mitigate this, a number of datasets representing different time periods and target platforms are often used in experimental work that are representative of different architectures as well as collection dates and methods. We follow this same practice in our work later.

Moreover, as well as evaluating a model during its development, its performance should be regularly monitored over time after it has been deployed in the wild. Recent work has shown that changes in the data distribution (known as *concept drift*) can result in a degradation in performance across several domains [144, 109]. For example, in malware detection, adversaries may use newer techniques to develop malicious executables. To cope with this problem and have the model adapt to such changes, several approaches are available with differing costs. The approaches include classification with rejection [109, 25], online learning [62], incremental learning [14] and active learning [179]. If performance declines significantly, however, it may be necessary to completely retrain the model on more recent data or reevaluate the feature engineering stage and the classifier design. Additionally, concept drift may also be considered at the stage where the dataset is split into training, validation, and test sets [19]. For example, using *temporal* splits of the dataset to produce training and test data (rather than random splits) may prevent results from being inflated owing to changes

in data distribution brought about by the constantly developing nature of malware [168, 167]. Nevertheless, the purpose of our work in this dissertation is to particularly investigate the robustness of defenses in relation to adversarial attacks. Hence, we do not focus on the aspect of concept drift, as it would be difficult to assess whether classifiers underperform due to concept drift or because of a lack of robustness towards attacks. The exploration of the relationship between concept drift and adversarial attacks is beyond the scope of our work. This is, in fact, cited as one of our future research directions.

In addition, there are well-established guidelines concerning experiments with malware and the deployment of ML-based malware detection systems that should be considered [183]. These pertain to various aspects such as the correctness of the dataset (e.g., how it is balanced across classes during training), the transparency (e.g., how malware samples are selected), and the safety (e.g., how real-world harm is avoided when conducting experiments). The purpose of these guidelines is to enhance the safety and security of malware detection systems. This is particularly important in light of the threat that more recent attack vectors pose to evade ML-based malware detection.

2.2 Attacking ML Models

In this section, we discuss the security of ML-based classification and detection systems by delving deeper into *adversarial machine learning*. An adversarial attack on an ML model occurs when an attacker manipulates input data at some point in the learning pipeline with the intention of causing the model to misclassify data [162]. Common types of adversarial machine learning attacks include *evasion attacks* that are performed at *test-time* [207, 32]; *poisoning attacks* that are performed at *training-time* [31]; and *privacy attacks* that include model stealing and membership inference attacks [196].

In our work, we focus on *evasion attacks* against machine learning classifiers at test-time, which are sometimes referred to as *inference-time attacks* or *integrity attacks* [162]. In such attacks, attackers generate an *adversarial example* by perturbing the features of an input sample in a particular manner to traverse the decision boundary of a classifier and induce a misclassification. Since the introduction of *adversarial examples* in work by Szegedy et al. [207], several evasion attack algorithms have been presented (e.g., [87, 134, 50]), and, in response, several defenses against them have been proposed (e.g., [161, 214, 227]), fueling another front in the cybersecurity arms race.

Prior to providing a detailed description of how evasion attacks against ML models operate and generate adversarial examples, we first describe common threat models from the relevant literature.

2.2.1 Threat Models

In the context of cybersecurity, it is essential to discuss threat models as they define the scope of attacks, the attackers' intentions, and their capabilities. The threat model characterizes the applicability, strength, and cost of attacks and provides descriptions of the attacker's capability and knowledge. Hence, prior to discussing adversarial evasion attacks in detail, it is imperative to lay some groundwork with regard to threat modeling.

In this dissertation, we focus on the same well-established threat models concerning the evasion of malware detection classifiers. Feature-based ML malware detection is a domain that has been widely explored in previous work [192, 229, 204, 88, 90, 32]. Although there are several ways to conceptualize a threat model, we primarily follow the widely-used formulation presented by Biggio et al. [32] and Laskov et al. [118], which has been revisited (and further clarified) in recent work by Apruzzese et al. [17]. With this formulation, attackers are distinguished by their level of *knowledge* and their ability to act on this knowledge (i.e., their *capability*). There are three main scenarios:

White-box/Adaptive (Perfect Knowledge)

In this scenario, the attacker has *complete and perfect knowledge* of the target model. This may include precisely what type of model the target model is, the hyperparameters used during its construction, access to the training dataset, and other information about the features used. While white-box access may be difficult to obtain in practical terms, it remains a valid threat scenario and is not completely unrealistic. For example, ML models can be trained in remote datacenters and then deployed on smartphones. Through reverse engineering, an attacker may be able to recover the model's internals from a smartphone and therefore achieve white-box access to the trained model [100, 162, 101]. The *adaptive attacker* scenario has also been presented in recent studies evaluating defenses against adversarial attacks [46, 215] as an additional characterization of the white-box scenario. As the attacker has comprehensive information about the target model, they can tailor their attack in order to evade the defense in a specific, targeted, and calculated manner. The adaptive attacker scenario can be used to enhance evaluations, following one of Kerckhoff's security principles [113], which is that a system should be secure even if the attacker knows everything about it (except any secret keys).

Gray-box (Limited Knowledge)

In the gray-box scenario, the attacker has *limited knowledge* pertaining to the target model. The gray-box attacker scenario may represent an insider threat or one where model data has been leaked. The knowledge could be limited information about the dataset used, knowledge of the feature representation as well as the statistical representation of the features across the dataset, or some other information about the target model. Any of this information could be used to improve an attack. With this limited knowledge, the attacker has an advantage over attackers with *zero-knowledge* but a significant knowledge gap as compared with a more knowledgeable attacker under the white-box scenario.

Black-box (Zero Knowledge)

In the black-box scenario, the attacker can only observe the predicted outputs for their queries. They have no additional knowledge about the target model, which represents a common threat model found in cybersecurity (such as when attackers target remotely-hosted systems). For instance, an attacker attempting to breach a network has limited access to the details of an intrusion detection system but may monitor how the target system reacts to network events. In cryptography, a common formulation of the black-box scenario is the *oracle*, where an attacker submits queries to the machine learning model as a black-box and examines the response received

[162, 17]. This is especially pertinent nowadays due to the ubiquity of Machine-Learning-as-a-Service platforms (MLaaS) (e.g., [198, 12, 61]). Regardless of the attacker’s knowledge level, however, one must presume that at least some information about the target model may be gleaned from published materials, literature, or the application domain itself [28].

2.2.2 Evasion Attacks Against ML Models

Having enumerated the common threat models, we now examine how evasion attacks against ML models are carried out. Evasion attacks are conducted in the feature-space of ML models, where the attacker perturbs the features of an input sample in a particular manner. The goal is to have the model produce a specific output that is different from the actual class, thereby evading the target model. For example, an image of a panda — which undoubtedly looks like a panda to the human eye — may be modified in such a way that it is misclassified by an ML model as another animal (such as a gibbon) [87]. In multi-class settings³, evasion attacks can be *targeted*, where the attacker aims to obtain a prediction for a specific class. Meanwhile, in an *untargeted* attack, the attacker only wants to induce a misclassification. As malware detection is a binary classification problem, targeted and untargeted attacks are equivalent since there are only two classes: positive and negative (malware and benign respectively). Thus, an attack against ML-based malware detection aims to have a truly malicious executable misclassified as benign [90], after being crafted in a calculated and deliberate manner to evade detection.

In domains such as image recognition, the feature-space is *continuous*. To conduct an attack in such domains, the continuous values in the feature vector (X , representing the image’s pixels) are perturbed in a particular manner to generate an adversarial example (X'). As portrayed in Figure 2.2, the adversarial example X' is designed so that it can cross the decision boundary, while remaining as close as possible to its original counterpart, which is controlled by various constraints (which we discuss later). The specific attack algorithm that is used and the attacker’s knowledge level dictate how the perturbations are performed in order to generate X' . Typically, though, small perturbations are applied (e.g., 0.01 is added to a particular feature value) until either X' successfully produces the desired predicted output from the target model or the resources are exhausted (e.g., possible perturbations), leading to attack failure.

Most attack algorithms can be classified as either *gradient-based* or *gradient-free* [216, 56]. Gradient-based attacks use information about the target model’s loss function with respect to input data for generating adversarial examples. Several gradient-based attacks have been introduced in recent years, such as the Fast Gradient Sign Method (FGSM) [87, 70], Projected Gradient Descent (PGD) [134], and Basic Iterative Method (BIM) [116]. Typically, these attacks compute the loss function with respect to input data and use the gradient information as a guide for making suitable perturbations that will cause a misclassification. However, importantly, these attacks require direct, unrestricted white-box access to the target model in order to operate. Though, as gradient-based attacks have direct access to the target model, they are typically faster and more effective than gradient-free attacks. Gradient-free attacks use other methods to achieve evasion, such as gradient-free optimization, iterative

³Multi-class settings are those where the output of the ML model can be one of at least three classes. Malware detection is a binary classification setting, with only two possible classes (i.e., outputs from a classifier): benign or malware.

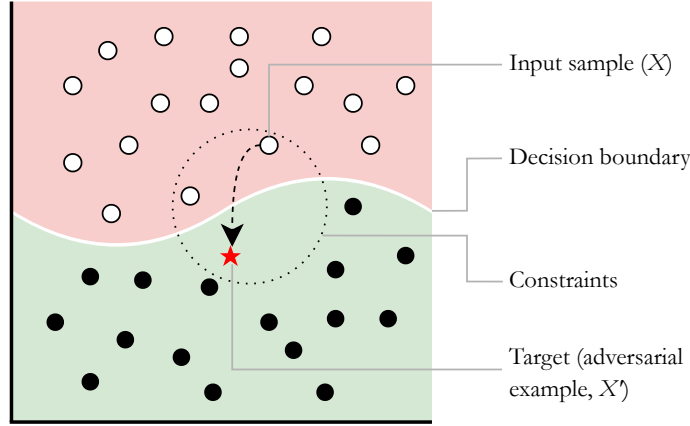


Figure 2.2: Overview of an evasion attack, illustrating the path followed to generate an adversarial example ($\star$). The dotted circle is a representation of constraints to limit the differences between the original sample X and the adversarial example X' .

perturbations and queries, evolutionary algorithms, or even gradient and decision boundary *estimation*. This type of attack is typically used in scenarios where the attacker has reduced or no information about the target model except the predicted output. A number of gradient-free attacks have been presented in recent years, such as HopSkipJump [50], ZOO [51] and Boundary attack [36].

Whether an attacker uses gradient-based or gradient-free attacks, they must ensure that there is some similarity between the original sample X and the adversarial example X' . To achieve this, a restriction on the degree of perturbations is normally enforced by using L -norms in some domains. For this, L_2 (the Euclidean distance between X and X') and L_∞ (the maximum pixel change in images) are most common [90, 134, 45]. If a successful adversarial example cannot be generated given the distance constraints, the attack is considered a failure.

Importantly, however, the majority of attacks introduced in recent years are designed to cater to the constraints of domains that use L -norms, but not the constraints of the malware detection domain for generating adversarial examples. L -norms cannot represent the key constraints of adversarial example generation in our domain and are therefore less applicable [181, 32], as we explain next. Therefore, it is particularly challenging to use these attacks as-is against ML-based malware detection classifiers. Rather, the attacks must be used as part of a wider strategy with additional steps to cater to the domain-specific constraints. This can limit the effectiveness of attacks when applied to this domain [182].

2.2.3 Evasion Attacks Against ML-based Malware Detection

Essential to the creation of any malware sample — whether it is an adversarial example or not — is that its malicious behavior and functionality remain intact. Recall that the feature vector X in ML-based malware detection is a binary feature vector that represents the presence or absence of different characteristics of the original software executable. Therefore, for example, a feature representing an API call in X (e.g., `CreateFile()`) cannot be perturbed as a continuous value by an attack when generating an adversarial example. That is, it

cannot be perturbed like: `CreateFile()` $+0.05$; rather, a completely different feature must be used [182] that offers the *same functionality*. This greatly increases the complexity of generating adversarial examples in this domain. Hence, when generating adversarial examples in the malware detection domain, the original *malicious functionality* of the software executable must be preserved, and the values in the feature vector must be *discrete*. Together, these two feature-space constraints offer a lower bound of functionality preservation, without the need for testing examples in a sandbox environment.

As a result, Figure 2.3 shows the generalized attack pipeline for generating adversarial examples in this domain, demonstrating the increased complexity. This shows that attack algorithms for other domains are used to generate adversarial examples that do not consider the constraints of this domain. Then, after an adversarial example is generated, it is ensured that the values in the feature vector are discrete and that all perturbations are valid according to pre-defined functionality preservation constraints in the feature-space (e.g., verifying that a feature required to achieve some functionality has not been removed). Meanwhile, in other attack pipelines, only those perturbations are initially applied that will not contravene the constraints of the domain, though this can increase costs and produce fewer adversarial examples [174]. In both cases, the valid perturbations are determined by consulting industry documentation for the target platform and prior work.

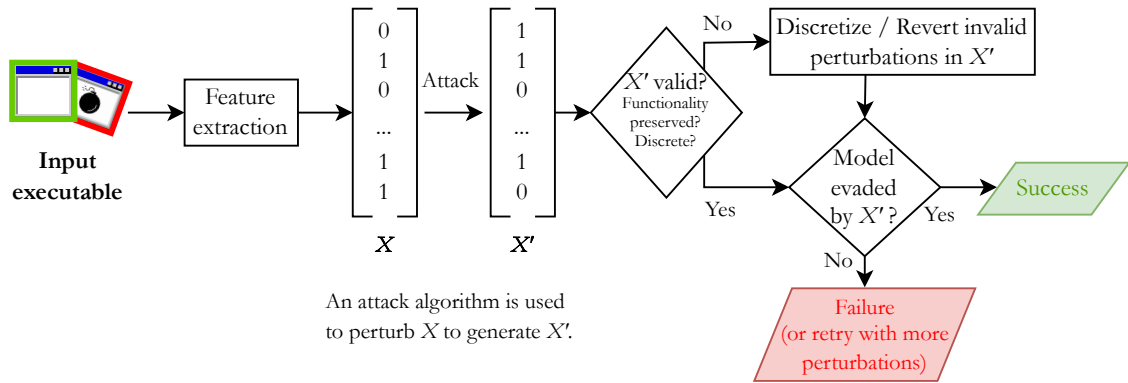


Figure 2.3: Overview of evasion attack pipeline in the malware detection domain.

The perturbations that can be applied to feature vectors to generate adversarial examples in this domain are either *feature addition* or *feature removal*. This contrasts with the continuous (i.e., non-discrete) modifications to the values in the feature vector that are used in other domains (e.g., $+0.01$ to a feature value). With feature addition, a value in the feature vector is perturbed from 0 to 1, so that a feature that was previously not present in X is added in X' . Meanwhile, feature removal perturbs a value from 1 to 0, thereby removing the feature. This is more complex and can only be performed when using certain datasets, as there is a greater risk of breaking malicious functionality. Importantly, although a generated adversarial example X' may represent a new malware sample in the feature-space, the perturbations used to construct it can be mapped to the problem-space to some extent. For example, feature addition can be achieved by adding dead-code or no-op calls within the code itself, or by using opaque predicates [146, 168]. Feature removal can be performed by rewriting the

dexcode in Android or by encrypting API calls and network addresses (thereby removing the features but retaining the functionality).

The purpose of the feature-space constraints is to offer a lower bound of functionality preservation for work that remains in the feature-space like ours (e.g., [192, 88]). In summary, to generate adversarial examples in the malware detection domain, the constraints are as follows (as based on prior work [181, 182, 67, 89, 168, 75, 193, 201, 90]):

- The *malicious functionality* of the original input sample X must be *preserved* in the generated adversarial example X' .
- The feature values in X' *must be discrete*. That is, each feature value must represent the presence or absence of a particular feature with a 1 or 0.

Based on these constraints, we can mathematically formulate the goal of attackers who try to evade ML-based malware detection models. Suppose we have the target model F and $F: X \in \{0, 1\}^M$ and we have some function $checkFunctionality()$ to check the malicious functionality of an input sample X . This goal can then be summarized as:

$$checkFunctionality(X) = checkFunctionality(X'); F(X) = 1; F(X') = 0 \quad (2.4)$$

In other words, the attacker's goal is to generate an adversarial example X' to evade the classifier F that results in a benign prediction from F . However, the original input sample X and the adversarial example X' must be functionally-equivalent in their malicious behavior. Thus, X' must retain the malicious behavior of X .

2.2.4 Transferability of Adversarial Examples

An interesting property of adversarial examples is that those developed for one model may evade other models because of weaknesses shared by classifiers. This is known as the *transferability* of adversarial examples [207]. This means that even if classifiers have different architectures or were trained on different datasets, an adversarial example generated for one model may *transfer* to another model and evade it [207, 157], allowing attackers to conduct generalized attacks even with limited knowledge of the target model. We explain later in Section 2.2.5.2 that an attacker can harness this property of adversarial examples to attack models in scenarios where they have no direct access to them [157]. As an attacker may be unable to apply attacks directly against the target model due to weaker capabilities, they can apply white-box attacks against a substitute model (which is an estimation of the target model) in anticipation that any generated adversarial examples will *transfer* to the target model. In other words, a black-box attack can be transformed into a semi white-box attack [157]. Through transferability, adversarial examples can also be reused against different models, thereby reducing the costs and resources involved in conducting an attack against several models [162, 157].

2.2.5 Attack Strategies Under Attacker Scenarios

After describing how evasion attacks work fundamentally in the feature-space and having enumerated three common attacker scenarios and threat models, we next discuss how attacks are conducted practically under each attacker scenario. This is driven primarily by the attacker’s knowledge and capability.

2.2.5.1 White-box Attacks (Perfect Knowledge)

In the white-box threat model, attackers have a high degree of information about the target model that allows them to conduct more effective and evasive attacks [162, 46]. In fact, while white-box access may be difficult to obtain, it is a valid and realistic attacker scenario in this domain. A significant number of attacks presented in the last few years assume white-box access to the target model, such as the Fast Gradient Sign Method [87], Projected Gradient Descent [134], Jacobian Saliency Map Approach [160], Basic Iterative Method [116], among others. These attacks employ techniques that harness direct access to the target model (e.g., access to gradient information and parameters) to guide the input sample across the decision boundary. This can include which features to perturb, which perturbations to apply to those features, and the level of perturbation needed, with a constant view of how the model responds during each attack iteration. Most of these attacks can be classified as gradient-based attacks that utilize the gradient of the target model’s loss function with respect to input data to control the perturbations.

However, the majority of white-box attacks are designed for other domains and therefore have no regard for the constraints of the malware detection domain. Therefore, after applying such a white-box attack to a malware input sample X , the feature vector of the generated adversarial example (X') must be *discretized*. Following this, any invalid perturbations must be reverted, such as those that are contrary to the permitted perturbations for the dataset, in order to preserve functionality. X' must then be tested to ensure that it is still adversarial for the target model, as features used to cross the decision boundary may have been reverted. By following this procedure, the constraints related to the generation of adversarial examples in the malware detection domain can be adhered to in order to achieve a lower bound of functionality preservation in the feature-space, without the need for testing in a sandbox environment.

2.2.5.2 Gray-box & Black-box Attacks (Limited & Zero Knowledge)

Conversely, in the gray-box scenario, the attacker has some knowledge and information about the target model, but not to the degree of an attacker with complete knowledge. Meanwhile, under the black-box scenario, the attacker can only rely on the predicted outputs from the target model to understand how the attack is progressing. In both of these scenarios, the attacker must treat the target model as an *oracle* that returns the predicted output and nothing beyond this⁴. This is representative of a growing number of MLaaS and remotely-hosted services where users do not have direct access to the model. This means that attackers under the gray-box and black-box threat models must rely on *transferability* and *query attack strategies* instead, as they do not have direct access to the model. Nevertheless, they may be able to build approximations of the

⁴Although some work (e.g., [181, 182]) assumes that the oracle may also provide confidence scores, we do not consider this in our work because it is a relaxation of the overall scenario as it provides attackers with more information.

target model or, at the very least, obtain some information about it through literature and the application domain [28]. Next, we describe these attack strategies in detail.

Transferability Attack Strategy. Even if the attacker does not have direct access to the target model, attacks can still be performed using white-box and gradient-based techniques. Due to the *transferability* property of adversarial examples (see Section 2.2.4), adversarial examples developed for one model may evade another because of weaknesses shared by classifiers [207]. Therefore, in a *transferability attack*, the attacker constructs a substitute model that is an approximation of the target model, as shown in Figure 2.4. This attack relies on the transferability property of adversarial examples to hold between the target model and the substitute model. Commonly, for this purpose, a deep neural network (DNN) is trained as the substitute model [157, 181, 36].

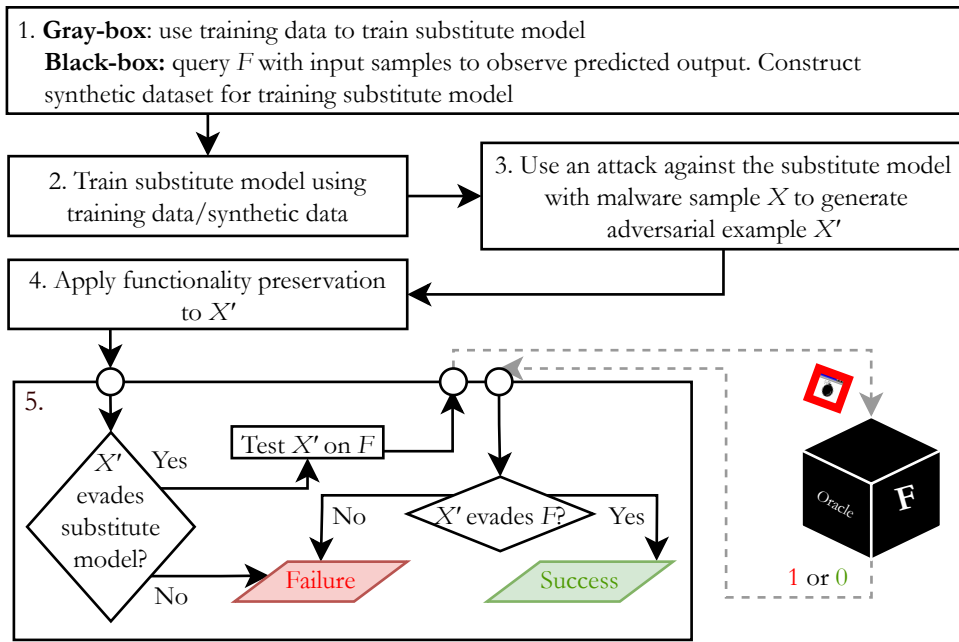


Figure 2.4: Overview of transferability attack pipeline for generating adversarial examples in the ML-based malware detection domain.

The procedure for constructing a substitute model differs under different threat models and attacker scenarios. Under the black-box scenario, the attacker has no capabilities beyond observing the predictions for their queries. Therefore, they can develop a synthetic dataset, which is an estimation of the input-output relations of the target model [157, 131, 175]. For this, the attacker can query the target model with input samples from both classes (benign and malware) and record the resulting predictions. The input-output relation between the queries and the target model's predicted outputs forms the synthetic dataset, which is then used to train a substitute model that acts as an approximation of the target model. Meanwhile, in the gray-box scenario, the attacker may try different techniques to develop a substitute model depending on their exact capabilities. For example, the attacker may have access to the training data of the target model, which can be used to train the substitute model [175].

In both scenarios, Figure 2.4 describes that once a substitute model is constructed, it is attacked using white-box attacks (e.g., FGSM [87, 70], PGD [134], BIM [116]) (see White-box Attacks in Section 2.2.5.1). The attacker anticipates that the generated adversarial examples will then transfer to the target model. Transferability attack strategies have been proven to work well in practice across different domains such as image recognition (e.g., [157, 131]) and malware detection [175], with different strategies involving a single substitute model [157], multiple substitute models [131] and diverse substitute models with additional techniques to increase the evasion rate [175].

Query Attack Strategy. Alternatively, query attacks generate an adversarial example by iteratively perturbing an input sample towards the desired class through feedback received from the target model [181, 50]. This is an example of a gradient-free attack, which is typically conducted in a completely black-box manner without any knowledge about the target model besides the predicted outputs. Figure 2.5 shows that by observing how the predicted output changes based on small perturbations to the input sample, the attacker can understand how the attack is progressing. In a gray-box scenario, the attacker may be able to use their increased knowledge to further guide the attack. For example, this could involve using information to determine which features are most likely to cause traversal of the decision boundary.

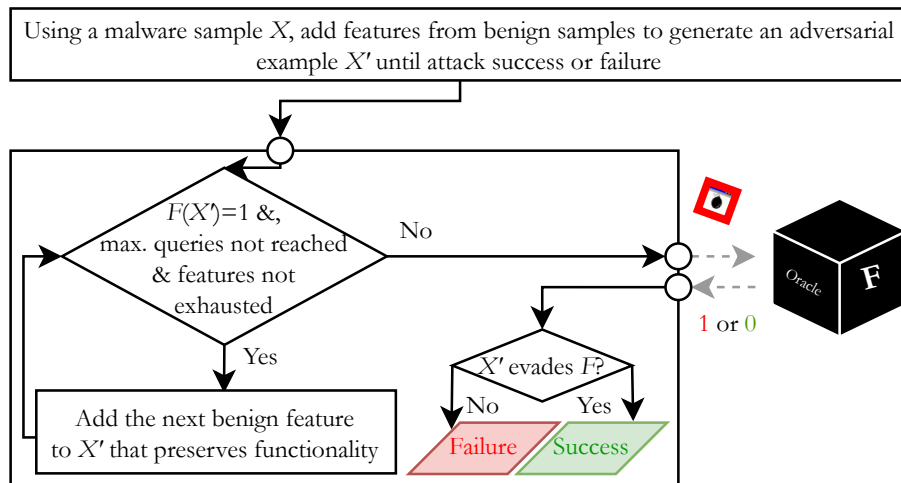


Figure 2.5: Overview of query attack pipeline for generating adversarial examples in the ML-based malware detection domain.

In other domains, several query attacks have been introduced that perform gradient and decision boundary *estimation* to perturb a continuous feature-space [52, 36, 50, 102, 129, 26], with recent attacks such as Hop-SkipJump [50] being particularly effective. However, in the malware detection domain, malicious functionality must be preserved, and discrete feature vectors must be used when generating adversarial examples [90, 181, 182]. Therefore, these attacks are generally ineffective as they pay no regard to these constraints, making changes to feature values in a non-discrete manner.

To this end, recent work in the malware detection domain has proposed software transplantation-based techniques to conduct a query attack [168, 182, 229, 175]. As shown in Figure 2.5, this involves perturbing

a malware sample with benign “donor” features to cross the decision boundary. This can be conducted in a scenario with less [182, 229] or more information [168] about the target model, which can be used to determine how to perturb features. As benign features are transplanted to the malware sample, it begins to exhibit more benign traits, which should cause a misclassification. Overall, this allows malware samples to cross the decision boundary of the oracle while catering to the constraints of this domain. An important factor to consider when conducting a query attack is limiting the number of queries to the oracle. This is because adversarial behavior as part of a query attack may be detected if queries are anomalous or too similar [52].

Universal Adversarial Perturbations. Recent research has also identified Universal Adversarial Perturbations (UAPs) as a cost-effective technique for producing adversarial examples [143, 117]. With UAPs, a single set of perturbations can be applied to multiple malware samples to generate adversarial examples. That is, sets of perturbations that are known to cause malware input samples to cross the decision boundary are reused across several malware samples to generate adversarial examples. If an attacker has knowledge of which sets of perturbations are universal, they can reuse them across several input samples in an attack, thereby reducing attack costs.

2.2.6 Evaluating Attack Performance

To evaluate the performance of attacks, several metrics and techniques have been developed that are applicable regardless of the attack strategy, scenario, or threat model used. In fact, these metrics can allow for the characterization and systematic comparison of different types of attack strategies (and the robustness of models, which we discuss in the next section). Recall that in our domain, a successful attack is an instance where the target model produces a *benign* prediction for a truly malicious input sample. A primary measure of attack performance is the *evasion rate*. This measures the number of attack attempts that are successful at evading the target model over the total number of attempts [207, 87]. A significant evasion rate can be an indicator of a stronger attack (or a weaker, non-robust model).

Moreover, a target model may exhibit behavior such that an adversarial example may evade the target model *sometimes*, due to its predictive nature (see Section 2.3.2 later). Therefore, the *repeat evasion rate (RER)* is an additional metric for evaluating attacks and the robustness of models [175]. This measures the number of times that an adversarial example evades the target model out of a defined number of attempts. Another metric to consider is the *cost* of achieving evasion. For example, the number of queries required to evade the target model can be an indication of the effectiveness of an attack [182, 157, 168]; a higher cost may imply a weaker attack. However, it should not be relied on as the only metric for measuring attack efficiency or effectiveness [17]. Furthermore, some work [174–176, 191, 46] has used standard machine learning metrics to evaluate attacks against models such as accuracy, F1, AUC, and false positive rate (FPR). By comparing a model’s performance on test sets with varying proportions of adversarial examples, a better understanding of the model’s performance under different system conditions can be derived. For example, a model’s accuracy may decrease as the adversarial intensity (i.e., the number of adversarial examples in a test set) increases.

These metrics and performance measures, when taken as a whole, enable us to derive a systematic comparison between various attack strategies and, more importantly, various defense systems. For instance, a

defense system that demonstrates higher accuracy, a lower false positive rate (FPR), and a lower evasion rate is significantly more reliable and desirable.

In the next section, we will delve into the various defensive approaches that can be utilized to protect ML models from adversarial attacks.

2.3 Defending ML Models

It is necessary to increase the adversarial robustness of prediction models to defend against the threat posed by adversarial ML attacks, while also maintaining sound performance on clean (i.e., legitimate) data. At the time of writing this dissertation, the development of defenses against adversarial attacks has proven to be a thriving research topic; hence, an exhaustive survey of all defenses against adversarial attacks is not within the scope of this work. Therefore, in the following sections, we will focus on the main concepts and well-known defensive approaches that have been proposed in recent years.

Broadly, defenses against adversarial attacks can be placed in one of the following three categories [28]: *model robustness*, where an individual model is made robust (i.e., a single-model defense); *ensemble defenses*, where a group of models is used to return a prediction; and *attack detection*, where a system tries to detect adversarial queries. Across these categories, defenses and techniques such as adversarial training (e.g., [207, 214]), gradient-based approaches (e.g., [161, 214]), feature-based approaches (e.g., [227, 221]), randomization-based approaches (e.g., [226]), moving target defenses (e.g., [58, 191]), and stateful defenses (e.g., [52, 130]) have all offered varying degrees of protection and robustness, with primary application and testing in the image recognition domain. Nevertheless, in the ongoing cybersecurity arms race, it has been found that the vast majority of defenses are ineffective against attacks that are subsequently-proposed. Several studies have highlighted the ineffectiveness of most defenses against adversarial ML in domains such as image recognition [46, 194, 21, 160, 162, 87], network systems, website fingerprinting and more [9, 195, 92, 148, 187]. This is primarily because of their static and fixed natures, their inability to cope with newer attacks, and their general ineffectiveness against stronger adversaries. Recent research (e.g., [169, 88, 122, 125]) has also applied these defensive approaches to ML-based malware detection models and demonstrated their general ineffectiveness in the malware detection domain. Importantly, however, a number of these studies have limitations, as they either do not adequately consider the constraints of the domain when generating adversarial examples to test defenses or they do not comprehensively evaluate different defense approaches, making it difficult to make systematic comparisons between them.

Furthermore, standards for evaluating proposed defenses have also been defined in recent work [46, 44, 19]. These aim to define core evaluation procedures such as testing using a variety of attacks (i.e., both transferability and query attacks), considering different attacker scenarios and threat models, assuming that an attacker has white-box access when interacting with the defense, and testing with multiple datasets. In addition, emphasis has been placed on evaluating the performance of defenses using a clean dataset (i.e., a dataset that does not include adversarial examples). This is because the performance on clean data may be degraded when techniques to increase adversarial robustness are introduced [44]. Moreover, the importance of considering the adaptive

attack scenario has also been stressed, as it allows for an evaluation of a defense at the peak level of threat when an attacker is aware of the defensive mechanisms in place [46, 215].

We next introduce defenses proposed in recent work in order to provide some context for the work presented later in the dissertation. Recall that some defenses are *single-model*, where only an individual model is made robust, while *ensemble defenses* use multiple ML models to thwart adversarial attacks. We also provide insight into *attack detection* through stateful defenses.

2.3.1 Single-Model Defenses (Model Robustness)

In the case of single-model defenses, an individual ML model (typically a deep neural network, or DNN) is made robust [175]. These are also known as *model robustness defenses* [28]. Importantly, these typically operate in a static manner and thus present adversaries with a fixed target to attack. For this reason, such defenses tend to show less robustness against newer attacks or stronger adversaries, and fail to prevent correlated or repeated attacks. However, there are some approaches that do show promise (e.g., adversarial training). We next provide some examples of well-known defenses in this category.

Gradient masking is an approach that is designed to prevent white-box attacks that use gradient information to generate adversarial examples [157]. When applied to a DNN, the idea is that the target model will not produce any useful gradients and therefore thwart attack attempts by misguiding the attack algorithm. However, recent work has shown that it is possible to recover usable gradients through a black-box attack [157], or to approximate the target model [21], which renders this defensive technique ineffective. Gradient masking is an example of *security through obscurity*, which itself has limited effectiveness from a wider cybersecurity perspective [113, 17]. Security through obscurity should only be used as part of *defense in depth* rather than as a standalone solution.

Several randomization-based defenses have also been proposed. Wang et al. have proposed *random feature nullification*, which reduces the attacker's chances of finding and using features that are *important* to the model when generating adversarial examples [221]. However, the effectiveness of this defense does not translate well from images to malware, with success only if the attacker makes random perturbations [202, 169, 174]. Xie et al. [226] have also proposed a *randomization-based defense* for the image recognition domain with transformations such as random resizing and padding, though this has been found ineffective in later work by Athalye et al. [21].

Another defense proposed by Papernot et al. is *defensive distillation* [161], which is principally designed for DNNs. In this approach, a primary DNN is trained with “hard” class labels from the training data in order to achieve maximum accuracy on the desired prediction task. Then, a second DNN is trained using the same input samples from the training data, but the hard labels are replaced with probabilities derived from the predictions of the DNN on those input samples. This second DNN is then used to make predictions for user queries, with the characteristic that it is able to generalize better. Though this defense has exhibited some success in the image recognition domain [161, 158], later work has suggested that it may not be much better than unprotected neural networks [43]. Furthermore, it has been found less effective when applied to the malware detection domain [202, 169, 88, 174].

Meanwhile, Szegedy et al. have proposed *adversarial training*, which remains an effective approach for protecting against adversarial attacks [207]. With adversarial training, adversarial examples are added to the training data to act as regularization in order to improve the model's ability to generalize. This has yielded success in several domains, including ML-based malware detection [169, 174]. However, adversarial training is not resistant to new attacks, meaning that a model must be trained on adversarial examples from newer attacks to stay up-to-date with the latest threats [86]. Adversarial training also introduces additional complexities, such as determining which model to choose as the base model, what to train on, and how much to train [174, 175]. Moreover, it has been found to cause overfitting in some instances [180] and worsen the performance of models on clean data.

A modification of adversarial training is *ensemble adversarial training*, as introduced by Tramer et al. [214]. In ensemble adversarial training, a model is trained on adversarial examples generated for itself as well as those generated for other models, thereby enhancing its ability to generalize even more. However, without frequent model updates and maintenance, this defense provides meager protection against unknown threats or newer attacks.

2.3.2 Ensemble Defenses

Ensemble defenses utilize several models as part of the decision-making process. They aim to outperform single models in terms of robustness, increased complexity for the attacker, variance reduction, prediction accuracy, and generalization [119]. Several ensemble defenses have been proposed, such as [55, 194, 232, 29, 30, 138]. However, recent work has shown that simply using an ensemble of models (including weak or non-robust models) is insufficient against the adversarial threat [97]. Hence, *strength in numbers* alone is not a valid defensive approach for dealing with adversarial attacks. Instead, the constituent models of the ensemble must be individually robust to some degree before they can be used in an ensemble [97].

Voting is a static ensemble defense that has been previously applied to the malware detection domain [55, 194, 232]. This is similar to a Multi-Variant Execution Environment (MVEE) where an input sample is fed into multiple models in order to assess divergence, and voting is used for the prediction [106, 105]. As Figure 2.6 shows, several constituent models are involved in making the final prediction. That is, each model makes a prediction for the input sample, and then a voting scheme is used to decide the final output, such as majority or veto voting. In majority voting, each constituent model makes a prediction, and the class with the most predictions is returned (e.g., if there are more “benign” predictions, the system returns this as its final output).

Meanwhile, in veto voting, if any constituent model returns a “malware” prediction, the entire system returns this prediction. In some cases, voting can cause a high false positive rate, such as with veto voting, because a single constituent model can incorrectly overrule predictions made by others. Additionally, a characteristic of voting is its increased time and memory costs for the defender, as all constituent models are involved in making a prediction [174, 175]. This is due to the fact that all constituent models must be retained in memory and predictions must be returned from each model before the final expected output of the voting system can be determined.

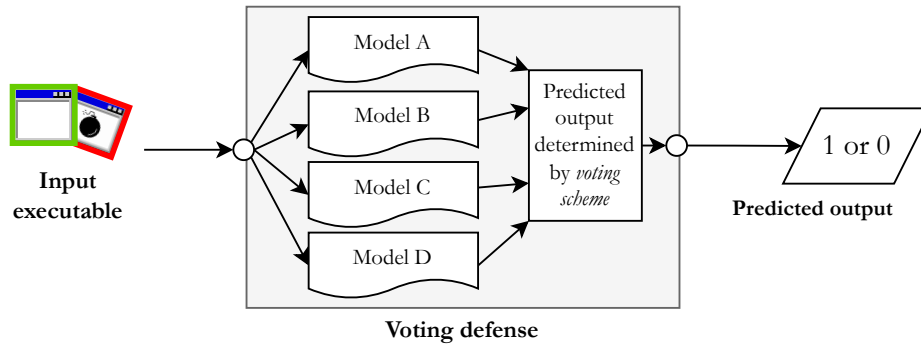


Figure 2.6: Overview of a voting defense. All constituent models make a prediction, with the final predicted output determined by majority or veto voting, or some other voting scheme.

Alternatively, *moving target defenses* (MTDs) have also been deployed in different security areas [58, 164, 209, 224, 108, 152] and applied in various fields, such as industrial control systems [163], network intrusion detection systems [220, 239], distributed systems [6], web applications [190], and cloud security [24]. MTDs seek to regularly change their configuration and randomize system components to increase uncertainty and make it more difficult for an attacker to understand how they behave [58] so as to not make themselves vulnerable as static, fixed targets to attackers. This behavior attempts to make fingerprinting and reconnaissance activities more difficult as well. Prior work has argued that defenses should exhibit some level of dynamism to thwart attacks, as most defenses that operate statically can be trivially evaded [86]. Hence, the key design principles of an MTD include “what to move”, “how to move” and “when to move” [58]. In the context of adversarial ML, this typically involves *moving* between several ML models using a particular method or strategy at some point (e.g., training-time, prediction-time), as shown in Figure 2.7. The constituent models are used to make predictions in an encapsulated manner, so that the attacker cannot ascertain how predictions are being produced. Thus, MTDs can be considered to belong to the category of *ensemble defenses*.

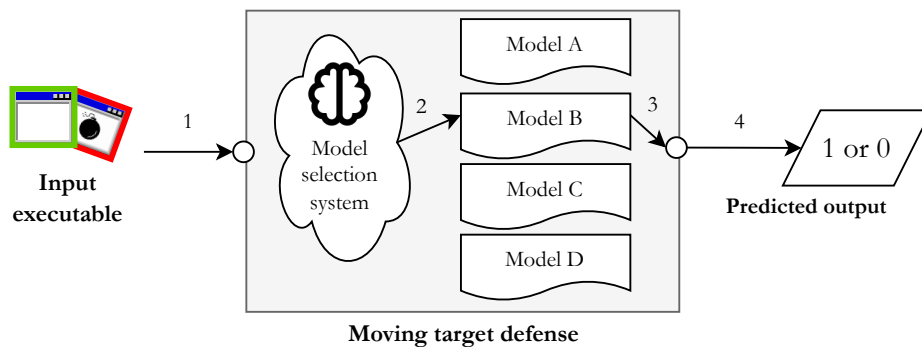


Figure 2.7: Overview of a moving target defense. The model selection system (which can be a randomized approach, algorithm, or strategy) determines the model to use at prediction-time.

For adversarial ML, MTDs can be categorized as either dynamic or hybrid [175]. For an input sample X , a dynamic MTD may return the prediction y at first, before returning the prediction y' later⁵, such that $y \neq y'$. Meanwhile, a hybrid MTD will continue to return the same prediction y until a specific condition is reached, causing it to alter itself (e.g., regenerating models if a query budget is reached). Then, the prediction for X may be y' such that $y \neq y'$. Subsequently, it will output y' until the condition is reached again. Contrarily, other ensemble defenses (such as voting [194]) are static, where the same prediction for an input is returned. The non-static predictive nature of MTDs introduces a further layer of complexity for attackers that is absent in other defenses.

Several MTDs have been recently proposed [58, 65, 191, 171, 200, 13, 184, 104, 174] to counter adversarial ML attacks. These defenses use different techniques that “move” the defense’s configuration with an element of dynamism at different stages of the prediction pipeline. A key benefit of using MTDs over single-model defenses and other approaches is that, in theory, attackers have it more difficult to understand how MTDs behave as they do not operate statically. Techniques for MTDs include model regeneration [200, 13], game-theoretic movement strategies [191, 184, 174], using adversarially-trained student models with different model selections [171, 13] and combinatorially-boosted ensembles [104]. Primarily, MTDs have been applied to and tested in the image recognition, demonstrating some level of robustness. However, as we explain and show later in this dissertation, most MTDs may not be effective in their current form to deal with attacks against ML-based malware detection models [175]. In particular, MTDs have not been applied to ML-based malware detection for defending against adversarial attacks. We provide a detailed discussion on MTDs later in Section 2.4.1.

2.3.3 Attack Detection & Stateful Defenses

The majority of defenses that we have discussed so far have largely been designed to defend models against white-box attacks and transferability attacks. These defenses aim to prevent the evasion of the model itself by an individual adversarial example rather than detect whether an attack is taking place, such as during query attacks. To specifically protect ML prediction models against query attacks (which are especially prevalent under the threat models where the attacker has less information), *stateful defenses* have been proposed.

ML prediction models — which include the aforementioned defenses — are typically *stateless* as they do not store or analyze the sequence of queries received. Recent work has stressed that systems must *monitor* queries to identify hazards such as adversarial attacks [99]. In fact, ML-based anomaly detection has been heralded as essential to detect the misuse of ML-based systems [37]. Thus, researchers have hypothesized that because query attacks iteratively perturb an input sample, they produce a sequence of abnormal queries compared with a sequence of legitimate queries. For example, adversarial queries may have abnormally high similarity to one another, or remain outside the distribution of previous queries (i.e., out-of-distribution, or OOD) [52, 130, 110, 112, 22, 52, 136, 39]. To this end, as shown in Figure 2.8, stateful defenses first retain the sequence of queries received by the system and then analyze them using different techniques and criteria to detect an attack. This is similar to intrusion detection systems and firewalls that monitor network activity to detect threats [35].

⁵For this reason, the Repeat Evasion Rate (RER) is a metric that can be used (see Section 2.2.6).

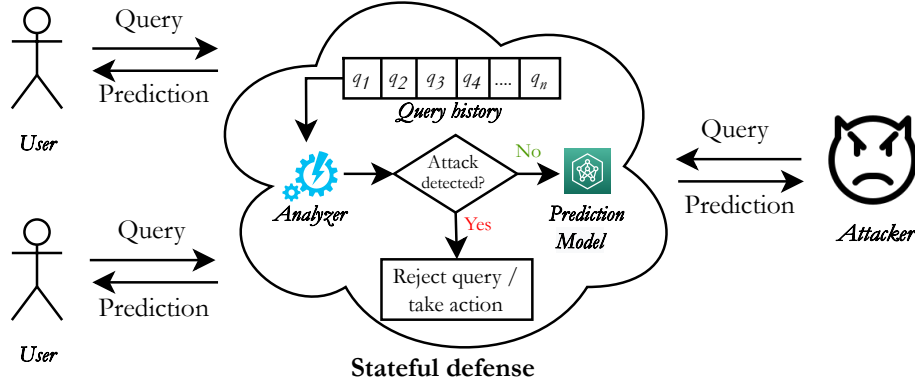


Figure 2.8: Overview of a stateful defense. Queries are stored and analyzed before reaching the prediction model. If an attack is detected, some action can be taken.

A number of stateful defenses have been proposed in recent years that use different methods for detecting adversarial queries [52, 110, 112, 234, 130]. Subsequent work has identified several limitations of these defenses. Importantly, however, stateful defenses have not been applied to ML-based malware detection before, therefore their effectiveness remains largely unknown. In fact, in the malware detection domain, the constraints for generating adversarial examples are not based on distance norms. Hence, as the methods to generate adversarial examples in this domain are inherently different, perhaps techniques beyond basic similarity and OOD detection are required to protect models. We present a further discussion on stateful defenses and their limitations later in Section 2.4.3.

Besides stateful defenses, other attack detection defenses have been proposed (e.g., [138, 89, 227, 188, 139, 69]). However, we primarily focus on stateful defenses in this dissertation, as this approach remains largely unexplored in the ML-based malware detection domain and shares similarities with other aspects of cybersecurity.

2.4 Focus of this Dissertation & Related Work

As remarked in the preceding sections, adversarial evasion attacks against ML-based malware detection have greatly widened the attack surface. Hence, thwarting such attacks is critical for the security of ML-based systems. In this endeavor, the remainder of this dissertation focuses on exploring novel methods to defend ML-based malware detection classifiers against adversarial evasion attacks. We propose two novel defensive methods based on approaches such as MTDs and stateful defenses — which have not been explored in this domain before. We thoroughly examine whether our defenses based on these approaches can provide any effective protection against attacks in comprehensive studies, where we compare and characterize different defenses for the first time. Due to the lack of prior application, the potential pitfalls of defenses based on these approaches in this domain remain unknown, and there are numerous unanswered questions⁶, such as: 1) Can

⁶We refer the reader to Section 1.3 for a detailed discussion about our research questions and aims.

a novel strategic moving target defense be developed to offer robustness against attacks, considering the various challenges related to developing this type of defense? 2) How do various defensive approaches (prior and our newly-proposed ones) compare with each other? 3) Can the performance and behavior of related defenses be characterized? Are there any key recommendations that can be derived for developing future defenses? 4) Based on one such recommendation, can a novel stateful defense reduce the evasion of adversarial query attacks in this domain? How does this compare with prior stateful defenses?

We aim to acquire a comprehensive and systematic understanding of the answers to such questions. Hence, we propose two new defensive approaches, one based on MTDs and another based on stateful defenses, which overcome the limitations of related work. We aim to understand how effective these approaches are at defending against adversarial attacks in ML-based malware detection and, if they are worth investigating further, what kinds of strategies and ways to design such defenses appear to be more effective for fostering future work. In order to provide further context for the work in this dissertation, we next present an overview of the research that is most related to it. We distinguish the sections of the related work by the subsequent chapters of this dissertation (which form our main contributions).

2.4.1 Strategic Defense Against Adversarial Attacks

In Chapter 3, we propose *StratDef*, which is a strategic defense system for the malware detection domain based on an MTD approach. Recall that MTDs routinely alter their configuration and randomize system components to increase uncertainty and make it harder to predict their behavior (see Section 2.3.2). The objective is to make it more challenging for attackers to perform meaningful reconnaissance and successful attacks [58], which will be rendered difficult as the target model will not be a static target. At the time of writing, several MTDs for adversarial ML have been proposed based on a variety of techniques (e.g., [191, 171, 13, 200]). Primarily, these MTDs vary the constituent models — which are mostly DNNs — used to make the prediction for a user’s query. A key limitation of prior MTD-based work is the lack of diversity in the constituent models, which makes transferability attacks more likely to succeed. In addition, as we will explain later, prior research on MTDs has failed to investigate a variety of important aspects connected to the design of MTDs. These include the challenges related to the systematic construction, selection, and strategic use of models to maximize adversarial robustness, which are core and fundamental. Additionally, to the best of our knowledge, there has been no prior work on applying or developing MTDs for ML-based malware detection. Prior MTDs are based on a number of different approaches, such as model regeneration [200, 13], game-theoretic movement strategies [191, 184, 171], using adversarially-trained student models with different model selections [171, 13], combinatorially-boosted ensembles [104], model diversity training [137], and others [5, 222]. In Table 2.1, we provide an overview of the most related literature for MTDs, covering the different approaches that have been employed in prior work. We next discuss each of these MTDs in detail.

	Song et al. (DeepMTD) [200]	Amich et al. (Morphence) [13]	Sengupta et al. (MTDeep) [191]	Qian et al. (EI-MTD) [171]	Roy et al. [184]	Martin et al. [137]	Izmailov et al. [104]
Uses diverse constituent models (i.e., non-DNN models)							
Model diversity training						✓	
Model regeneration	✓	✓					
Student-teacher models	✓	✓		✓			
Game-theoretic movement			✓	✓	✓		
Voting-based decision		✓					✓
Combinatorially-boosted ensembles							✓
Hybrid MTD/movement	✓	✓					✓
Dynamic MTD/movement			✓	✓	✓	✓	

Table 2.1: Overview of the most related literature for MTDs for adversarial ML covering the different approaches.

MTDs that use model regeneration function by regenerating the pool of models that make up the MTD, while the system itself is under deployment. Song et al. propose *DeepMTD* as one such MTD approach [200] where student DNNs are created by perturbing the weights of a base DNN followed by retraining. The student models are updated whenever the system is in an idle state. An input is sent to all student models, and the final prediction is determined by majority voting. That is, once all the constituent DNNs of the MTD have made a prediction, the class with the most votes counts as the output of the MTD. However, a key issue with this approach is that the model pool is only renewed when the system is in an idle state, which an attacker could prevent it from entering through constant interaction. This would make the MTD a static target, contradicting its very purpose. Meanwhile, Amich et al. propose *Morphence* [13], which generates student models from a base model by shuffling the weights of the base DNN. After this, a subset of the student DNNs is adversarially-trained. The rationale behind this approach is that the MTD will offer balanced performance on clean and adversarial inputs. The constituent models are regenerated when a query budget is exhausted (i.e., after a certain number of user queries) to keep the system moving. For a user’s query, the most confident model’s prediction is returned. As a side-effect, *Morphence* has the potential to increase the number of false positive predictions as the most confident model may not make the correct prediction (and thus behave like a veto voting system). Moreover, the dependence of *Morphence* on adversarial training can reduce accuracy on clean, legitimate data in some cases. Additionally, this approach only uses DNNs as constituent models, thus reducing model diversity and heterogeneity. Furthermore, it may be possible to discover the query budget, which could be used by an attacker to determine how this MTD moves.

Game-theoretic movement strategies have also been used in several MTDs to determine which constituent model should be used to serve the prediction. *MTDeep* proposed by Sengupta et al. [191] uses different DNN architectures to minimize the transferability of attacks. Using a Bayesian Stackelberg game, a model is chosen at prediction-time to make a prediction for an input sample. However, like *Morphence*, a key drawback of the *MTDeep* approach is that it only uses DNNs and is thus limited in its model heterogeneity, which increases the vulnerability to transferability attacks. Moreover, in some domains, *MTDeep* may produce only pure strategies

(as we find later in Chapter 4). Other MTDs based on game-theoretic approaches have also been proposed, such as the MTD proposed by Roy et al. [184] and *EI-MTD* by Qian et al. [171]. EI-MTD utilizes a Bayesian Stackelberg game to dynamically schedule student models in the edge intelligence domain. Using a base model from the cloud that is adversarially-trained, the student models are constructed through differential distillation and serve predictions on edge devices with limited resources. However, EI-MTD’s heavy dependence on adversarial training can decrease its accuracy on clean data, perhaps more than Morphence. This is because all of EI-MTD’s student models are adversarially-trained as opposed to a subset of them, as in the case of Morphence.

Model diversity training has been employed in an MTD approach proposed by Martin et al [137]. The key idea is to promote diversity among the constituent models of the MTD by using a regularization technique, which the researchers refer to as “adaptive diversity promoting”. Although, in principle, diversity should be a key characteristic of an MTD’s constituent models, the researchers themselves find that by employing a surrogate model attack, an attacker can evade the randomized output decision strategy employed by this MTD entirely. That is, the underlying method of creating diversity among constituent models used for thwarting transferability attacks is insufficient. Conversely, Izmailov et al. propose an MTD using *combinatorially-boosted ensembles* [104]. The idea is to derive different combinations of ML models, fuse (i.e., combine) their outputs, and then decide the final prediction using some method based on voting. However, there are several limitations to this approach. Firstly, the constituent models of this MTD are not diversified; rather, they have the same architectures but are initialized with random hyperparameters. Additionally, this MTD approach uses a voting mechanism in order to derive the final conclusion, which has its own limitations relating to lack of dynamic behavior, increased time and memory costs, as well as false positives (see Section 2.3.2). Furthermore, the system can refuse to classify some queries, which limits the quality of service for users.

An overarching limitation of all these existing MTD approaches is that they do not consider various key factors, such as the challenges related to the systematic construction, selection, and strategic use of models to maximize adversarial robustness. This limits their dynamic behavior and makes them more susceptible to attacks, as an ensemble of models cannot be trivially put together to introduce robustness [97]. For example, most prior MTD-based work only uses small ensembles of models, consisting mainly of DNNs as the constituent models of the MTD. Additionally, these approaches only vary the DNNs in their training procedures, thereby limiting model heterogeneity at prediction-time, which can render them more vulnerable to transferability attacks. In contrast, in Chapter 3, we present a new MTD with our StratDef approach. We explore how and to what degree the model selection procedure should be heuristically-driven to promote key aspects such as heterogeneity, redundancy, and to minimize the effect of transferability of adversarial examples across the constituent models of an MTD. Moreover, unlike other defenses, StratDef can give consideration to information available about its operating environment to provide an adapted and tailored response based on the current threat level. A major advantage of this is that the defender can use their resources efficiently, reducing time and storage costs, while still having a system with sound performance. In addition, the behavior of StratDef can be optimized based on the circumstances; for instance, if a certain metric needs to be prioritized, the model selection can be modified accordingly.

Furthermore, an important aspect of our work on a strategic MTD is the comparison between it and prior defenses for adversarial ML. We focus on the consideration of functionality in the feature-space when generating adversarial examples that are used to evaluate and compare a range of defenses. In relation to this, there is a research avenue that has compared defenses for adversarial ML in malware detection. However, some of these evaluations do not consider the constraints of the domain sufficiently for generating adversarial examples (e.g., [169, 88, 90, 53, 208]). For example, some work [169, 88] uses L_p norms to constrain the generation of adversarial examples rather than evaluating which feature perturbations may affect the functionality. This would prevent adversarial examples from representing malware samples in the feature-space. Some other work does not consider constraints at all [89]. Meanwhile, prior work that does consider the domain's constraints is often restricted by the scope of their comparisons among defenses, concentrating mostly on variants of adversarial training and ensemble methods [122, 126, 177, 121, 235, 64, 127]. Because of this, it is more challenging to make a direct comparison between various defensive approaches. Recent work by Severi et al. [192] does take domain constraints into consideration and compares proposed solutions against a range of defenses. However, this research pertains to poisoning attacks rather than evasion attacks, which are the focus of this dissertation.

2.4.2 Effectiveness of Moving Target Defenses Against Adversarial Attacks

In Chapter 4, we study the effectiveness of several recent MTDs applied to counter adversarial ML attacks in the malware detection domain. Based on our findings, we present key recommendations for spurring future work on the development of effective MTDs for adversarial attacks in ML-based malware detection.

Past research on MTDs [58, 65, 174, 191, 171, 200, 13, 184, 104, 5, 137] has not evaluated them under different threat models, nor compared MTDs with each other, nor offered practical recommendations for developing effective MTDs based on an evaluation of recent MTDs. That is, the performance and behavior of MTDs have not been characterized through experimental work prior, to the best of our knowledge. It is essential to determine if MTDs are a worthwhile research direction to protect against adversarial ML attacks and, if so, what kinds of approaches for designing MTDs appear to be more effective. Our work in Chapter 4 is the first evaluation of its kind and is part of a series of evaluations of defenses against adversarial ML attacks. Generally related to the effectiveness of defenses, prior work has demonstrated that most single-model defenses are ineffective at dealing with adversarial examples [169, 194, 21, 160, 162, 87]. For ensemble methods, work by He et al. examining ensemble defenses [97] has shown that ensembles of weak defenses are insufficient against adversarial ML. In their work, using three ensemble defense configurations (not based on MTDs), they show that independent defense mechanisms that are not intended to work together fail to offer notable robustness against attacks. Meanwhile, in our work in Chapter 4, we are the first to conduct a comparative evaluation and characterization of MTDs.

To conduct the evaluation of MTDs, we use a combination of prior attack strategies and novel ones that we propose for increasing evasion against different models, especially MTDs. Prior work by Papernot et al. [157] explored methods for attacking remotely-hosted single-model classifiers based on a black-box threat model introduced in [118]. They propose a successful attack that involves harnessing the transferability property of

adversarial examples by developing an estimate of the target model (as a substitute model) and attacking it to generate adversarial examples. The motivation behind this is that since the generated adversarial examples are produced for the substitute model (which is an estimation of the target model), some of them may transfer to and evade the target model. Later in our work, we refer to this strategy as the Single DNN strategy. Building on this attack strategy, Liu et al. [131] have hypothesized that if an adversarial example can evade several substitute models, it is more likely to transfer to other models too, such as the target model. In their ensemble-based attack, they demonstrate that by using multiple substitute DNNs (i.e., an ensemble of DNN substitute models), the evasion rate against the target model can be improved. We later refer to this as the Ensemble DNN strategy. As we show later in Chapter 4, our transferability attack strategy improves on all these approaches. Rather than using just DNNs as substitute models, we use an ensemble of diverse substitute models (from different model families) and validate the transferability of adversarial examples across the substitute models before evaluating them on the target model. The goal of this strategy is to ensure that adversarial examples are highly-transferable so as to increase the chance of evasion against as many of the constituent models of an MTD as possible.

Prior work has also developed query attacks (e.g., [52, 36, 50, 182, 102, 129, 168, 26]) to generate adversarial examples using techniques such as gradient and decision boundary estimation, though primarily for images. These are designed to be used in scenarios where the attacker has considerably less information about the target model, such as the black-box threat model. However, most of these query attacks have little capacity to deal with discrete features and functionality preservation, which are core constraints of the malware detection domain when generating adversarial examples [182]. To this end, the query attack strategy that we present later in Chapter 4 is inspired by Rosenberg et al.'s query attack for ML-based malware detection [182], which uses a software transplantation-based approach. However, our threat model considers even less information about the target model under the black-box condition, as we assume no access to prediction confidence scores or usage of sliding windows. Therefore, we use a modified version of their decision-based attack without these assumptions. When evaluating MTDs, it is important to consider more restricted threat models where the attacker has little information about the target model, since the purpose of MTDs is to encapsulate their own behavior and reveal as little as possible about how predictions are produced.

A key characteristic of MTDs is that they try to make fingerprinting and reconnaissance activities more difficult through regular movement. In relation to prior work on fingerprinting and reconnaissance (which we also cover in Chapter 4), several studies on adversarial ML have been conducted in other domains, such as network systems and website fingerprinting [9, 195, 92, 148, 187], though not for MTDs, to the best of our knowledge. Other work has explored model stealing, which is a different adversarial aim from ours [213, 219, 32, 28]. For instance, [213] shows how an attacker can obtain parameters from remote classifiers with partial knowledge of models. Nonetheless, we show later that the hyperparameters of MTDs and details about their behavior can be retrieved in a black-box setting.

Another core contribution of our work in Chapter 4 is the concrete recommendations for designing MTDs based on our comprehensive evaluation. Generally, prior work has not provided sound recommendations for constructing MTDs based on empirical data. Although Martin et al. [137] make some suggestions about MTDs, these are quite different from ours. For example, we do not recommend randomization or using only deep learning models in an MTD. As we show later in Chapter 4, MTDs relying solely on DNNs as their

constituent models exhibit poorer adversarial robustness, especially against transferability attacks. We also show that MTDs that use dynamic and strategized movement mechanisms offer better performance against attacks through a comparison of several MTDs, rather than basing recommendations on a less effective MTD approach as in [137].

2.4.3 Stateful Defense Against Adversarial Query Attacks

In Chapter 5, we present MalProtect, which is a stateful defense for the malware detection domain. In a comprehensive evaluation, we show that MalProtect outperforms existing stateful defenses, especially under the peak adversarial threat.

ML prediction models and most defensive approaches for adversarial ML are typically *stateless*, as they do not store or analyze the sequence of queries received. Upon receiving a new query, a stateless defense will make a prediction based on the particular method that it is built with. For example, a model may be resilient to an adversarial query (and therefore return a malware prediction) perhaps because it has been adversarially-trained. However, without access to previous user queries, such models and defenses are unable to detect ongoing attacks or a sequence of adversarial queries. Recall that with query attacks, an attacker iteratively perturbs an input sample based on feedback from the target model, which includes the predicted output. Using the feedback, the perturbations to the input sample are tuned across the queries to observe how the target model responds until, eventually, the sample is classified differently [50, 182].

It has been hypothesized that because query attacks iteratively perturb an input sample, they produce a sequence of abnormal queries compared with a sequence of legitimate queries. Therefore, *stateful defenses* put the query received by the system in the context of prior queries. They analyze the sequence of queries made to the system in order to detect attacks, such as analyzing the similarity of feature vectors representing an image. Several stateful defenses for adversarial ML have been proposed in recent years [52, 110, 112, 22, 234, 130, 136]. Principally, these rely on distance-based similarity detection and out-of-distribution (OOD) detection approaches to detect adversarial queries. As we discuss next, there are a number of limitations to these defenses, such as insufficient detection mechanisms, poor maintenance of the query history, and weaknesses regarding the use of account-based detection, among others. In Table 2.2, we provide an overview of the most related literature for stateful defenses, covering the different approaches that have been employed in prior work. We next provide detail about each of these approaches, discussing how they detect attacks and any key limitations.

	Chen et al. (SD) [52]	Li et al. (Blacklight) [130]	Azmoodeh et al. [136]	Kariyappa et al. [111]	Atli et al. [22]	Juuti et al. (PRADA) [110]	Zhang et al. (SEAT) [234]
Maintains query history	✓	✓	✓			✓	✓
Distance-based / Similarity detection	✓	✓	✓			✓	✓
Account-based defense	✓		✓			✓	
Out-of-distribution (OOD) detection				✓	✓	✓	✓
OOD based on training data				✓	✓		

Table 2.2: Overview of the most related literature for stateful defenses and attack detection for adversarial ML, covering the different approaches.

Stateful defenses that use similarity detection operate under the assumption that adversarial queries are abnormally similar to each other. If there are pairs of queries that have a distance below some threshold (that is, they exhibit a degree of similarity), it can be assumed that an attack is taking place. However, this is also a fundamental limitation of this approach, as an attacker could carefully design queries to be dissimilar, thereby evading detection [99]. Additionally, in domains such as malware detection, an attacker can trivially increase the distance between queries by adding unused features, as we show later in Chapter 5. An example of a stateful defense based on similarity detection is Chen et al.’s *Stateful Detection approach (SD)*. This is an account-based defense that maps a user’s query to a feature vector and measures the average distance between it and its k -nearest neighbors. If this is below a threshold, an attack is detected, which resets the query history and cancels the user’s account. However, this can be defeated by a Sybil attack, where an attacker uses multiple accounts in the attack [71, 228]. Moreover, inspecting queries per-account limits the detection scope [130]. An alternative approach proposed by Li et al. is *Blacklight* [130]. This stateful defense measures the similarity of queries for all users using the L_2 distance (i.e., Euclidean distance), which is a type of L_p norm [45]. An attack is detected if the Euclidean distance between several queries (represented by hashes) is below the threshold. Like SD, however, Blacklight can be evaded by attacks that intentionally use dissimilar queries or through query-blinding attacks, which are specific to the image recognition domain [234]. Other distance-based stateful defenses have also been proposed, such as Azmoodeh et al.’s stateful detection approach for NLP models [136]. However, this is also an account-based defense that relies on similarity detection methods that can be evaded by knowledgeable attackers, as acknowledged by the researchers themselves. Moreover, these threshold-based defenses require the actual selection of the threshold, which in and of itself may prove to be difficult.

Meanwhile, several out-of-distribution (OOD) detectors have been proposed to prevent attacks and detect adversarial examples [39]. These examine whether a query belongs to the distribution of the target model’s training data [112, 22]. Since an adversarial query is unlikely to belong to this distribution, OOD queries can be flagged and an attack can be detected [39]. When OOD queries are detected, Kariyappa et al. propose returning inconsistent labels to those detected OOD queries [112]. However, an attacker may be able to construct adversarial queries that remain within the required distribution [234, 233], thereby evading detection. Atli et al.’s OOD approach [22] instead uses a binary classifier based on residual networks that

aims to distinguish whether a query is from the training data of the underlying prediction model. However, the authors acknowledge in their work that a sufficiently knowledgeable and capable attacker may be able to go virtually undetected if they are able to craft queries in a specific manner to remain within the required distribution.

An alternative OOD approach is *PRADA*, which has been proposed by Juuti et al. [110]. This is a defense against model extraction attacks, which is a problem related to evasion attacks. Interestingly, prior work has found that *PRADA* can also be used to detect evasion attacks [52]. *PRADA* is based on the assumption that the L_2 distance among non-adversarial queries follows a normal distribution, which it monitors using the Shapiro-Wilk normality test among the queries received by the system. When a query appears to fall outside of this distribution, an attack is detected. Although *PRADA* has shown promise at detecting attacks, the manipulation of the query distribution has been found to evade it [52, 233]. Additionally, like the other stateful defenses that we discuss, *PRADA* is an account-based defense, which would mean that a capable attacker could also conduct a Sybil attack to evade this defense [130]. Zhang et al. have proposed *SEAT* [234], which is also a stateful defense for model extraction attacks and is based on perceptual similarity measures. However, key drawbacks of this defense are that it is only compatible with DNN prediction models and, more importantly, that it has not been found to be successful in detecting evasion attacks.

Besides the limitations of prior stateful defenses, it is important to note that ML-based malware detection differs greatly from other domains. Here, attackers must adhere to stricter constraints regarding the discrete representation of feature vectors and the maintenance of malicious functionality when generating adversarial examples [181, 32]. Consequently, attackers employ strategies to produce adversarial examples that are distinct from those used in other domains [182, 168]. For example, modifying a query by altering some “pixels” is inadequate for generating adversarial examples in ML-based malware detection [182]. Therefore, stateful defenses — such as those based on similarity detection — which offer robustness when applied to other domains, may be unsuccessful when applied to the malware detection domain as the detection methods are insufficient. As we demonstrate later in Chapter 5, methods beyond basic similarity and OOD detection are required. To this end, we later propose *MalProtect*, which uses techniques such as analyzing the autoencoder loss of queries, examining the distribution of enabled and shared features across queries, and more, to achieve superior detection performance.

2.5 Conclusion

In this chapter, we presented the essential background information necessary for appreciating the work in the following chapters of this dissertation.

We began by introducing machine learning and the fundamental ideas behind ML-based malware detection. We presented an explanation of how feature-based machine learning classifiers are used to detect malware, which is an example of supervised learning and, more specifically, a classification problem. We highlighted that ML classifiers should only be deployed if they are accurate, offer a low false positive rate, and perform well with respect to other metrics too, lest a disruptive system with poor quality of service be deployed.

Next, we discussed the security of ML-based classification systems. We explained the threats posed by adversarial machine learning, highlighting how an attacker can deceive and evade an ML model. As threat models are a core aspect of systems security, we enumerated common ones found in the literature that cover different attacker knowledge and capabilities. Based on these, we explained how different attack strategies could be employed for evading ML models, such as white-box, transferability, and query attack strategies to generate adversarial examples. We also emphasized that the malware detection domain has more constraints when generating adversarial examples compared with other domains. These relate to the discretization of features and functionality preservation in the feature-space. We discussed how, despite several adversarial attacks having been proposed in the literature, many do not pay heed to these constraints. Then, we highlighted the necessity of making ML models robust against these attacks. With reference to prior work, we examined how a number of defenses against adversarial ML attacks had been proposed but found to be generally ineffective in several domains, such as image recognition and malware detection.

In this dissertation, we therefore propose novel defenses to defend ML-based malware detection classifiers against adversarial attacks. Prior work has failed to develop, or even apply, any strategic or stateful defenses against adversarial attacks in the malware detection domain, while other approaches have a number of key limitations. It remains to be seen whether new defenses based on these approaches can offer robustness. Moreover, there has been little effort in evaluating different types of defenses in this domain comprehensively or in considering the constraints of the domain properly when investigating defenses in prior work. Therefore, we also aim to characterize and study different types of defenses for adversarial ML in this domain.

In the next chapter, we propose a novel moving target defense against adversarial attacks based on a strategic approach. We investigate how a strategized MTD may provide better protection against adversarial attacks in ML-based malware detection by addressing the limitations of prior work and exploring key aspects of an MTD, such as what, how, and when it should adapt itself. We also present a comprehensive evaluation of several models and other defenses to provide context for the performance of our defense, where we correctly consider the constraints of the domain under various threat models. This allows us to draw a systematic comparison between the different defensive approaches.

CHAPTER 3

STRATEGIC DEFENSE AGAINST ADVERSARIAL ATTACKS

In this chapter, we propose a novel moving target defense against adversarial attacks based on a strategic approach called *StratDef*. In contrast to prior work, StratDef selects the best models to use in an MTD in a way that is both dynamic and strategic. We provide an entire framework for MTDs that focuses on the fundamental aspects of their design, such as model construction, model selection, and optimizer selection, which prior work has not considered in as much detail. This helps not only to increase accuracy and the level of uncertainty for attackers, but it reduces the transferability of adversarial examples. In a comprehensive study, we show that StratDef outperforms other types of defenses for adversarial ML even under the highest levels of adversarial threat. The work in this chapter relates to OBJ1 and OBJ2, where we aim to propose a strategic MTD and evaluate it in the context of other defenses for adversarial ML.

3.1 Introduction

As we discussed in Chapter 2, a number of defenses for adversarial ML have been proposed based on a variety of approaches that primarily aim to increase the robustness of individual models. However, the effectiveness of these defenses has been found to be limited for a variety of reasons, including that they tend to remain static, fixed targets [86] that are especially susceptible to repeated and correlated attacks. While complete security is difficult to achieve, a system’s goal should be to control the attack surface as much as possible in order to thwart adversarial attacks.

In this regard, there have been meager efforts at approaching the problem of defending against adversarial ML from a dynamic and strategic perspective. For this purpose, in other areas of cybersecurity, MTDs are employed that continually vary themselves to increase the uncertainty and complexity for the attacker, thereby making reconnaissance and targeted attacks less successful [150, 58]. Although prior work (e.g., [191, 171, 13, 200]) has proposed MTDs for defending against adversarial ML with some degree of success, as we discussed in Section 2.4.1, they have a number of limitations. For example, prior MTD approaches have not explored the various key factors that we argue should be considered when building MTDs, such as the challenges related to the systematic construction, selection, and strategic use of models to maximize adversarial robustness. By not considering these factors, previously proposed MTDs may lack dynamic behavior, which makes them more susceptible to attacks as constituent models put together in an ensemble cannot alone provide adequate protection [97]. The process of designing MTDs is non-trivial and must deal with selecting

the constituent models of the ensemble and then how to strategically use them. For example, prior MTD-based work has only used small ensembles of models, consisting mainly of DNNs as the constituent models, and has varied these DNNs only in their training procedure. These models have then been used in a non-strategic and even non-dynamic manner, leading to a number of limitations (see Section 2.4.1 for a detailed discussion on the limitations of prior MTDs). Additionally, and importantly, MTDs have not been applied to defending against adversarial attacks in ML-based malware detection.

Therefore, in this chapter, we present a new MTD approach named StratDef. We advance the state of the art by providing a method for constructing a strategic MTD that embraces the key areas of model construction, model selection, and optimizer selection. Rather than plainly utilizing an MTD approach with a group of models, we provide an entire framework for generating models, selecting those models systematically, and producing reliable strategies to use those models. This allows StratDef to offer accurate predictions against legitimate inputs while also defending against adversarial examples. In this work, we explore how and to what degree the model selection procedure should be heuristically-driven to promote key aspects such as heterogeneity, redundancy, and to minimize the effect of transferability of adversarial examples across models. Moreover, unlike other defenses, StratDef can give consideration to any information available about its operating environment to provide an adapted and tailored response based on the current threat level. A benefit of this is that it enables efficient use of resources, potentially reducing time and storage costs for the defender, while still having a system with sound performance.

In this chapter, we conduct a detailed evaluation of several prior defenses and compare them with StratDef to provide some context for the performance of our strategic MTD. In our evaluation, we correctly consider the constraints and characteristics of the ML-based malware detection domain regarding the generation of adversarial examples, unlike some prior work (e.g., [169, 88, 90]). The results show that we can increase accuracy by 50+% in the most adverse conditions for both Android and Windows malware under a variety of attacker scenarios with StratDef compared with other models and defenses. Later, in Chapter 4, we study the effectiveness of several recent MTDs and compare StratDef's performance with them in the first evaluation of its kind.

We next introduce the threat model used in this chapter, followed by a detailed description of our defense, StratDef.

3.2 Threat Model

As discussed in Section 2.2.1, feature-based ML malware detection is a domain that has been widely explored in previous work (e.g., [192, 229, 204, 88, 90, 32]). We therefore focus on the well-established threat model concerning the evasion of such malware detection classifiers. That is, the attacker's overall objective is to evade an ML-based malware detection classifier so that their malicious input is classified as benign.

3.2.1 Application Model

We follow the application model presented in Section 2.1.3. Namely, the target model of the attack is a binary classification model that predicts whether an input sample belongs to the benign or malware class. Recall that to construct such a classifier for malware detection, executables are represented as binary feature vectors. For this purpose, datasets often provide a comprehensive set of extracted features from real-world executables. With these datasets and features $1 \dots M$, we can construct a vector X for each input sample such that $X \in \{0, 1\}^M$. $X_i = 1$ indicates the presence of feature i and $X_i = 0$ indicates its absence. We use the feature vectors and associated class labels to construct binary classification models for malware detection, similar to the one shown in Figure 2.1.

3.2.2 Attacker's Goal

The attacker's goal is to generate adversarial examples to evade a malware detection classifier. Suppose we have a classifier F , where $F: X \in \{0, 1\}^M$ and a function *checkFunctionality()* to check the functionality of an input sample. Then, this goal can be summarized as:

$$\text{checkFunctionality}(X) = \text{checkFunctionality}(X'); F(X) = 1; F(X') = 0 \quad (3.1)$$

In other words, the attacker wants to generate an adversarial example X' that is functionally-equivalent to the original malware sample X , but is predicted as *benign* by F . Recall that we use 0 to represent the benign class and 1 for the malware class.

3.2.3 Attacker Capabilities & Knowledge

For the majority of this chapter, we model all attackers who interact with StratDef in a gray-box setting with *limited knowledge* about the target model, like previous work [162, 118, 185, 28]. In our threat model, attackers have access to the same training data as the target model and knowledge of the feature representation. However, attackers have no knowledge of the parameters, configurations, or constituent models of StratDef nor any other evaluated defenses. Therefore, they must train substitute models using the training data and attack them, in the expectation that the generated adversarial examples will transfer to the target model [207, 118, 157]. This is based on the well-established idea that adversarial examples for different models can be used to evade the target model [207].

Furthermore, we use different scenarios involving attacker capabilities and attack intensities with the goal of studying and evaluating the performance of StratDef under different *threat levels*, like in prior work [44, 46, 28, 203, 192, 168]. Attackers' capabilities may differ in the behavior, the strength and intensity of their attacks, their ability to generate adversarial examples, and more. Later, in Section 3.4.4, we describe precisely how we model different profiles for representing attackers with different capabilities.

Nonetheless, for deployment, in the absence of any information about the operating environment, StratDef assumes the highest threat level, consisting of the most adverse environment and the strongest attacker. However, if there is information about the operating environment and/or the attackers within it (e.g., through cyber-

threat intelligence [197, 238] or situational awareness), StratDef can use it to provide a more targeted defensive approach. Therefore, in our evaluation (see Section 3.5 later), we show how StratDef performs against different attacker scenarios and intensities to show the whole range of its capabilities. Nonetheless, for the comparison with other defenses later, we focus on the strongest attacker, as this is the default scenario when no information is available about the attacker or environment.

Additionally, we evaluate StratDef’s performance against a black-box attacker with *zero knowledge*, as featured in previous work [102, 162, 157, 181, 159, 36, 52, 68, 75, 128]. This attacker only has access to the predictions of StratDef and no other knowledge. The attacker therefore performs a transferability attack, in which they construct a substitute model as an estimation of the target model by querying it systematically. The substitute model is attacked in the hope that any generated adversarial examples transfer to the target model [207].

In the following section, we provide a detailed description of our strategic moving target defense, StratDef.

3.3 StratDef

In this section, we firstly describe our strategic method, StratDef, at a high level and then provide details about each of its steps. StratDef embraces the three key design principles of an MTD: what to move, how to move, and when to move [58, 150, 222]. We provide a systematic and heuristic model selection system to determine *what to move* considering the current user type, the threat level, and performance metrics. With this method, StratDef’s behavior can be optimized according to the situation; for example, if a particular metric needs to be prioritized, the model selection can be adjusted accordingly (as we describe later).

As presented in the overview of StratDef in Figure 3.1, once models have been selected, we can strategize *how* they will be used. In cybersecurity, an MTD typically cycles through configurations during its deployment. Since StratDef makes predictions for legitimate and adversarial inputs, we use a strategy to choose a model at prediction-time, thereby strategically cycling through the models *when* it is time to move. We explore multiple methods to determine this strategy, ranging from uniform random probabilities, a game-theoretic approach based on Bayesian Stackelberg Games, and a strategic heuristically-driven approach.

Overall, StratDef provides an entire method for choosing the models to use in the ensemble and then how to use them at prediction-time to serve predictions to users. We next provide details for the key steps of StratDef, which are shown in Figure 3.1.

3.3.1 What to Move: Phase 1 — Model Generation

The first step is conducted offline (that is, at the development stage — see Figure 3.1). This consists of generating the models that are going to be considered when forming the StratDef ensemble. Let U refer to the set of candidate models considered by the defender for inclusion within the ensemble. These models can be constructed by the defender or taken from other work. U can contain vanilla models as well as models that already incorporate individual defenses (single-model defenses as discussed in Section 2.3.1, e.g., adversarial training [207], random feature nullification [221], SecSVM [69], defensive distillation [161]). In addition, U

can contain models of different families, such as decision trees, neural networks, random forests, or support vector machines, as well as models of the same family but with different parameters (e.g., neural networks trained adversarially but to different levels).

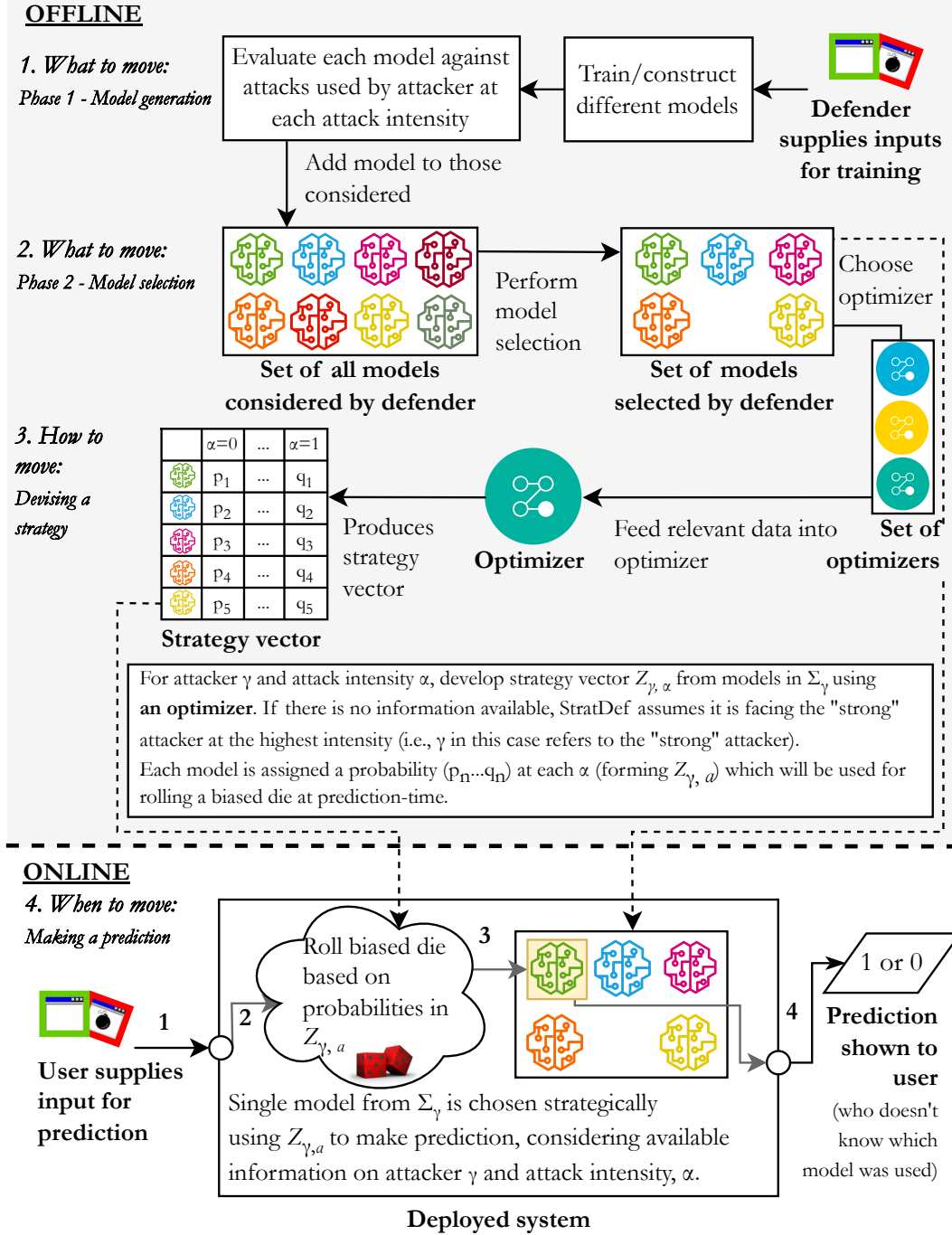


Figure 3.1: Overview of StratDef.

3.3.2 What to Move: Phase 2 — Model Selection

Continuing offline, the model selection procedure is performed to produce a strong, heterogeneous ensemble of models to be used. Recall that StratDef provides a tailored defense by considering the information it may have about its operating environment. That is, the model selection (and the strategy derived for using those selected models later) is tailored to the operating environment based on any available information about it. However, if no specific information is available, an environment with the highest threat level is assumed (i.e., the strongest attacker and highest attack intensity) with the model selection reflecting this. A benefit of having a tailored model selection (provided that information is available about the environment) is that the StratDef ensemble will be appropriate for the level of threat, potentially reducing the defender's costs (e.g., resources, time required for training, storage). For example, the costs of training highly-robust models could be avoided if StratDef is deployed in a less hostile environment where less robust models might suffice.

To the best of our knowledge, with our model selection approach, we are the first to offer a flexible method to select models systematically by considering model performance and threat levels. To achieve this model selection system, we *simulate* threat levels by generating adversarial examples and pooling them (we provide details in Sections 3.4.3 & 3.4.4 later on how the adversarial examples are generated and pooled). Each candidate model $F \in U$ is then evaluated under each threat level using several machine learning metrics. This allows us to aggregate the metrics into a *consideration score*, which encapsulates the performance of each candidate model at different threat levels.

A higher consideration score for a model indicates its better performance, which may increase its chances of being selected for the StratDef ensemble. The actual formula for the consideration score can vary based on the deployment needs and requirements. For example, a defender may be more interested in minimizing false positives over other metrics (see Section 3.4.2 later for the specific formula we use in our experimental evaluation). Therefore, in Equation 3.2, we provide a *general* formula for the consideration score:

$$S_{F,\gamma,\alpha} = \oplus(m_{F,\gamma,\alpha}^1, m_{F,\gamma,\alpha}^2, \dots, m_{F,\gamma,\alpha}^n) \quad (3.2)$$

$S_{F,\gamma,\alpha}$ refers to the consideration score of a candidate model $F \in U$ at attack intensity α against attacker γ . That is, a particular combination of the metrics chosen by the defender is considered (e.g., whether metrics are weighted, maximized, minimized, etc.). For the n considered metrics, $m_{F,\gamma,\alpha}$ refers to the metric m for the candidate model F at attack intensity α against attacker γ . The defender can choose the metrics and attribute weights to each in whatever manner they require as part of the consideration score. Depending on the situation, one may adjust the considered metrics or use different metrics altogether to produce a new model selection.

Once the consideration scores are produced, the candidate models are sorted in descending order by their consideration scores at each threat level (i.e., each attack intensity α and each attacker γ). Essentially, this procedure ensures that the candidate models can be sorted by their performance considering several ML metrics at different threat levels. This drives the model selection method for selecting which candidate models will be part of the StratDef ensemble. We explore two different model selection methods that use the consideration scores and other characteristics of the candidate models (and evaluate these methods later in Section 3.5):

- **Best selection method** — This method selects the best-performing models with the aim of maximizing performance across the considered metrics. For each attacker and attack intensity (i.e., each threat level), we select the k highest-scoring models out of all potential candidates in U . k is a hyperparameter and can be chosen by the defender according to their requirements and resources (e.g., a defender with greater resources may want to use more models). The models selected *may* be from the same model families, yet they demonstrate some robustness and are therefore chosen to deal with attacks.
- **Variety selection method** — This method aims to reduce transferability among the selected models by enforcing diversity in the model selection. The highest-scoring model from each model family in U is selected at each attack intensity against each attacker. The number of models selected per attack intensity is equal to the number of model families.

Once the selections are made according to the method, the model selections for each attack intensity α for each attacker γ are pooled together and represented by the set Σ_γ , so that it contains all the models selected across all attack intensities for the attacker γ . Recall that if no information is available about the operating environment, γ represents a strong attacker, with the most capable models selected to deal with this kind of environment. This model selection process offers a systematic yet flexible approach to the defender, allowing them to adjust the considered metrics, which would adapt StratDef to their needs. For example, the defender could modify the formulation of consideration score, the candidate models to be considered, or the model selection method itself.

3.3.3 How to Move: Devising a Strategy

So far, StratDef has selected an ensemble of models for each threat level (i.e., each attacker and attack intensity) that are considered most suitable according to various criteria. The models selected across the attack intensities for each attacker (γ) have then been pooled together into the set Σ_γ . The next step is to determine *how* these models will actually be used at prediction-time to serve predictions to users. For this, an optimizer is used to strategize how each model in Σ_γ (the models that were selected in the previous step) will be chosen at prediction-time by StratDef for making predictions. This step takes place offline and corresponds to choosing an optimizer for developing the movement strategy, as in Figure 3.1.

Each optimizer produces a global strategy vector Z_γ for an attacker γ using data about the models. The probability of choosing each model from Σ_γ at attack intensity α against attacker γ at prediction-time is contained within $Z_{\gamma,\alpha}$. This means that the strategy chooses from the most suitable models to make the prediction by adapting to the attack intensity and the attacker type (i.e., the threat level). With the probabilities in Z_γ , a biased die is thrown at prediction-time to select a model for making the prediction. In Table 3.1, we provide an example of a strategy vector, with each value in the strategy vector corresponding to the probability of choosing a model $\sigma \in \Sigma_\gamma$ at threat level α .

	$\alpha = 0$	$\alpha = 0.1$	...	$\alpha = 1$
σ_1	0	0.2	...	0.3
σ_2	1	0.4	...	0
...	...	...	...	...
$\sigma_{ \Sigma_\gamma }$	0	0	...	0.1

Table 3.1: An example of a global strategy vector Z_γ with the probabilities for each model $\sigma \in \Sigma_\gamma$ at each α against attacker γ .

In order to devise the movement strategies and the strategy vector Z_γ , we explore three optimizers and evaluate them later in Section 3.5. We next provide a description of how a strategy vector is produced by each optimizer that we consider in this work.

Game-Theoretic Optimizer (GT)

We can model the problem of adversarial examples as a min-max game between an attacker and a defender, following the well-established concept of *Security Games*. This has been successfully applied to various areas of both physical security and cybersecurity [164]. Specifically, in our problem setting, the attacker is trying to maximize the loss of the classifier, while the defender is trying to minimize it.

Hence, we model the interaction between the defender ($\mathcal{D}$) and the user — who can be either a legitimate user ($\mathcal{L}$) or an attacker (γ) — as a Bayesian Stackelberg Game [209]. The defender is the leader, and the user is the follower. The defender aims to maximize its expected reward over its switching strategy and the strategy vector played by the followers. We produce payoff matrices for each game between the defender and each user.

The game between the defender and the attacker is modelled as a constant-sum game like previous work in other domains [191, 171]. In prior related work, the utilities in game-theoretic formulations of adversarial ML have been based on the evasion rates of each attack (the attacker’s possible move) against each model (the defender’s possible move), but without any consideration for how many adversarial examples are produced. In the malware detection domain, attacks may be more or less effective because of domain-specific constraints. Additionally, a stronger attacker can generate a greater number of more evasive adversarial examples than a weaker attacker. Therefore, we instead use a *normalized evasion rate* as the utility value in the game-theoretic formulation in order to encapsulate information about the number of adversarial examples generated as well as the evasion rate. We provide details of the procedure below.

Let $\Omega_{\tau,S}$ represent the set of adversarial examples generated by an attack τ for some substitute model S that is used to generate adversarial examples (see Sections 3.4.3 & 3.4.4 later for implementation and experimental instances):

1. Evaluate each set of adversarial examples $\Omega_{\tau,S}$ against each selected model $F \in \Sigma_\gamma$ to obtain the evasion rate. That is, measure the proportion of adversarial examples from $\Omega_{\tau,S}$ that evade the model F .
2. Compute the normalized evasion rate ($R_{\tau,S,F}$) to reflect the evasive capability of the set $\Omega_{\tau,S}$ against model F . For this, multiply the number of adversarial examples in the set by the evasion rate and

normalize between 0 and 100. A constant-sum game ($= 100$) is the frequent setup for the game between an attacker and defender [191, 171].

3. Produce payoff matrices, where the defender is the row player, for each game by calculating rewards:
 - (a) For the constant-sum game between $\mathcal{D}$ and γ , the attacker's reward is equal to the normalized evasion rate $R_{\tau,S,F}$. The defender's reward, because it is a constant-sum game, is therefore equal to $100 - R_{\tau,S,F}$.
 - (b) For the game between $\mathcal{D}$ and $\mathcal{L}$, the reward for both players is equal to the accuracy of the model F on a clean test set (i.e., the defender's possible move).
4. Feed both payoff matrices into a Bayesian Stackelberg solver (such as [164, 93]) along with the attack intensities. This produces a strategy vector $Z_{\gamma,\alpha}$ containing the probability of playing (i.e., selecting) each model $F \in \Sigma_{\gamma}$ against attacker γ at attack intensity α as $p(F, \gamma, \alpha)$.

In the optimization problem, α is a hyperparameter modelled as a trade-off between accuracy on legitimate and adversarial inputs corresponding to the attack intensity. The optimization problem may result in a pure strategy (where only a single model is chosen for predictions) or a mixed strategy (where there is a choice between multiple models). A pure strategy can be produced when one of the models is more robust than others. At $\alpha = 0$, StratDef is only concerned with accuracy on legitimate inputs, and therefore a pure strategy of the most accurate model is produced.

Strategic Ranked Optimizer (Ranked)

With this optimizer approach, we use the consideration scores for each model in the set $\Sigma_{\gamma,\alpha}$ (i.e., the models selected for attacker γ at attack intensity α) to produce a strategy vector. At $\alpha = 0$, a pure strategy consisting of the most accurate model is produced. Meanwhile, for $\alpha > 0$, each model in the set is sorted by its consideration score. A rank is then assigned to each model in the sorted set, with the lowest-scoring model having a rank of 1. The rank increases as the model scores increase. Based on this, each model is assigned a probability in $Z_{\gamma,\alpha}$ as per Equation 3.3:

$$p(F, \gamma, \alpha) = \frac{r_{F,\gamma,\alpha}}{\sum_{G \in \Sigma_{\gamma,\alpha}} r_{G,\gamma,\alpha}} \quad (3.3)$$

$r_{F,\gamma,\alpha}$ is the rank of model F at attack intensity α against attacker γ . In other words, the probability of a model F being selected is its rank divided by the sum of all ranks. Therefore, we assign the highest probability of being selected to the highest-scoring model. In $Z_{\gamma,\alpha}$, a probability of 0 is assigned to models that are not in $\Sigma_{\gamma,\alpha}$. In other words, if a model was not selected at a particular attack intensity, it will have a probability of 0 in the strategy vector. This approach will always generate a mixed strategy at every attack intensity except $\alpha = 0$.

Uniform Random Strategy (URS)

This approach assigns a uniform probability to each model in Σ_γ and only acts as a baseline for comparing with the other approaches, as it is not expected to give the best performance. The uniform random strategy approach maximizes the uncertainty for the attacker with regard to the model that is selected at prediction-time. Thus, the probability is calculated according to Equation 3.4:

$$p(F, \gamma, \alpha) = \frac{1}{|\Sigma_\gamma|} \quad (3.4)$$

3.3.4 When to Move: Making a Prediction

After the offline generation and selection of the best models as well as the creation of the strategies to move between the selected models — that is, the strategy vector Z_γ to move between models in the set Σ_γ — StratDef is now ready to be deployed online and start making predictions. As per Figure 3.1, when a user requests a prediction, StratDef will choose a model from Σ_γ to pass the input to by rolling a biased die using the probabilities in the strategy vector $Z_{\gamma,\alpha}$ in real-time. The chosen model will actually make the prediction that will be returned to the user. In the absence of information about the threat level of the environment, StratDef will assume it is facing the strong attacker at the highest attack intensity. Because the actual model that is used by StratDef to make each prediction will be chosen dynamically, the user will have it difficult to know how predictions are being produced, let alone which model was used at prediction-time. Therefore, our hypothesis is that if the previous steps are performed systematically following our method, StratDef will offer sound and robust predictions, while revealing minimal information about itself.

Next, we show how StratDef performs better than existing defenses in the malware detection domain in the face of adversarial examples. In the following section, we provide details of the experimental setup we consider for the evaluation, together with details about how we generate adversarial examples.

3.4 Experimental Setup

3.4.1 Datasets

In Section 2.1.5, we discussed the challenges and considerations related to experimental work in security applications. In particular for malware detection, a well-known, general problem is the lack of publicly-available, up-to-date datasets that can be used in academic research, limiting the merits and conclusions of work in this domain [19, 20]. To mitigate this to some extent, in this dissertation, we use a number of datasets that cover different platforms and collection methods, which have been widely used in prior work (e.g., [223, 90, 66, 117, 168, 109, 23, 166]).

In this chapter, we use two datasets: one for Android and another for Windows. We use the DREBIN dataset for Android [18], which consists of 123,453 benign samples and 5,560 malware samples. There is a total of eight feature families consisting of extracted static features ranging from permissions, API calls, hardware requests and URL requests. To keep our dataset balanced, we use 5,560 samples from each class (benign

and malware), resulting in a total of 11,120 samples with 58,975 unique features. For Windows, we use the SLEIPNIR dataset [8], which offers a convenient binary feature-space allowing for a simpler comparison of results between the datasets. SLEIPNIR consists of 19,696 benign samples and 34,994 malware samples. The features of this dataset are derived from API calls in PE files parsed into a vector representation by the LIEF library [212, 8]. We use 19,696 samples from each class, resulting in a total of 39,392 samples with 22,761 unique features. Similar to recent publications [69, 90], and for completeness, we use the maximum features for each dataset. For our experimental evaluation, both datasets are transformed into a binary feature-space with each input sample parsed into a feature vector representation. The datasets are initially split using an 80:20 ratio for training and test data using the Pareto principle. After this, the training data is further split using an 80:20 ratio to produce training data and validation data. This effectively produces a 64:16:20 split, which is a technique that has been widely used before [133, 178, 120, 230, 73, 231].

We also consider the established guidelines for performing malware-related research [183] following the discussions in Section 2.1.5. For example, as the models in our evaluation decide whether an input sample is benign or malicious, it is crucial to retain benign samples in the datasets, and we do not need to strictly balance datasets over malware families. Rather, we balance datasets between the positive and negative classes (i.e., benign and malware) and select unique samples from each class to appear in the training and test sets randomly (without any chance of repetition) [169, 8, 181].

3.4.2 Training Models & Defenses

3.4.2.1 Other Models & Defenses

We compare StratDef with a range of other models and defenses. To construct these models, we use the scikit-learn [165], Keras [59] and Tensorflow [2] libraries. We construct four vanilla models (see Table 3.2 for architectures), which are the base models for defenses such as ensemble adversarial training [207, 214], defensive distillation [161], SecSVM [69], and random feature nullification [88].

Model	Parameters & Configuration
Decision Tree (DT)	max_depth=5, min_samples_leaf=1
Neural Network (NN)	4 fully-connected layers (128 (Relu), 64 (Relu), 32 (Relu), 2 (Softmax))
Random Forest (RF)	max_depth=100
Support Vector Machine (SVM)	LinearSVC with probability enabled

Table 3.2: Architectures of vanilla models.

We apply *defensive distillation* to the vanilla NN model. Meanwhile, *random feature nullification* is applied to all vanilla models, where we randomly nullify 10% of the features so that they are no longer considered by the models. For *SecSVM*, we use the vanilla SVM model as the base model and apply the robustness procedure following the original paper.

For *adversarial training*, we train the vanilla models with different-sized batches of adversarial examples (ranging from 0.1% to 25%) from those generated in the next section. For example, suppose the size of the training set is 2224 (which is equally split between benign and malware samples). Then, for a 0.05 model

variant (e.g., NN-AT-0.05), we select the adversarial examples (i.e., 5% of half the training set size) and add these to the training set. We then train the vanilla and SecSVM models to produce adversarially-trained models. Preliminary work shows that adversarially training beyond 25% increases time and storage costs as well as overfitting.

We also compare StratDef with the voting ensemble defense. **Voting** has been applied to other domains [222] and to the malware detection domain [194, 232]. This is similar to a Multi-Variant Execution Environment (MVEE) where an input sample is fed into multiple models in order to assess divergence, and majority voting is used for the prediction [106, 105]. We use the same constituent models for the voting defense as for StratDef (and thus the naming conventions are similar). We consider two voting approaches that have been tested in prior work [194, 232]: *majority voting* and *veto voting*. The better of the two approaches is compared with StratDef. Preliminary work shows that veto voting causes a higher false positive rate (FPR) in both datasets using the previously described test set — as high as 25% in DREBIN (see Figure 3.2). The higher FPR may be because the voting system is forced to accept any positive prediction from its constituent models, even if it is incorrect (hence, a false positive). Therefore, we focus on comparing StratDef with majority voting using the same model selections.

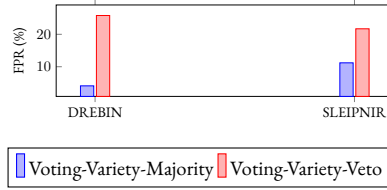


Figure 3.2: FPR of Voting-Variety-Majority & Voting-Variety-Veto against the strong attacker averaged across attack intensities. “Voting-Variety” refers to voting schemes using the Variety model selection (see Section 3.3.2).

3.4.2.2 StratDef

To construct different StratDef configurations, we follow the offline steps described in Section 3.3 to construct models and devise strategies. The candidate models are the individual models and defenses trained as described above (except voting). We aim to maximize the accuracy and robustness on input samples while minimizing false predictions. To achieve this, we use the formula in Equation 3.5 for the consideration scores, where we maximize accuracy (ACC), AUC, F1 and minimize FPR and false negative rate (FNR) across the threat levels.

$$S_{F,\gamma,\alpha} = ACC_{F,\gamma,\alpha} + F1_{F,\gamma,\alpha} + AUC_{F,\gamma,\alpha} - FPR_{F,\gamma,\alpha} - FNR_{F,\gamma,\alpha} \quad (3.5)$$

$S_{F,\gamma,\alpha}$ is the consideration score of the candidate model F at attack intensity α against attacker γ . The value of each metric for the candidate model F at attack intensity α against attacker γ is represented accordingly. We use all combinations of the Best (with $k = 5$) and Variety model selection methods with the three optimizers described in Section 3.3.3 to produce six StratDef configurations (see Appendix B.1 for example strategies developed by StratDef).

3.4.3 Generating Adversarial Examples

As discussed extensively in Section 2.2.3, when generating adversarial examples in the ML-based malware detection domain, it must be ensured that the feature vectors remain discrete and that the original malicious functionality is preserved. We achieve this by limiting the perturbations that can be applied to the feature vector, as a lower bound of functionality preservation. This ensures that adversarial examples remain close to realistic and functional malware without the need for testing in a sandbox environment. We therefore generate adversarial examples in the feature-space like previous work [123, 8, 90, 69].

3.4.3.1 Preliminaries

To recap from Section 2.2.3, there are two types of perturbations that can be applied to a feature vector to generate an adversarial example. Feature addition is where a value in a feature vector is modified from 0 to 1. In the problem-space, an attacker can achieve this in different ways, such as by adding dead code so that the feature vector representing the software changes, or by using opaque predicates [146, 168]. This has proven to work well to create adversarial malware, for instance in Windows [68]. Meanwhile, feature removal is where a value in a feature vector is modified from 1 to 0. This is a more complex operation, as there is a chance of removing features that affect the functionality [124, 3, 123, 168]. For Android apps, an attacker cannot remove features from the manifest file nor intent filter, and component names must be consistent. Furthermore, the S6 feature family of DREBIN is dependent on other feature families and cannot be removed. Therefore, the opportunities for feature removal lie in areas such as rewriting dexcode to achieve the same functionality, encrypting system/API calls and network addresses. For example, obfuscating API calls would allow those *features* to be removed (since they would then count as new features) even though the functionality would remain [123, 168].

Hence, for each dataset, we determine the allowed perturbations by consulting industry documentation and previous work [124, 3, 123, 8, 168, 117]. DREBIN allows for both feature addition and removal, with Appendix A.1 providing a summary of the allowed perturbations for each of the feature families. For SLEIP-NIR, we can only perform feature addition because of the encapsulation performed by the feature extraction mechanism of LIEF when the dataset was originally developed.

As we describe in the next section, we include a verification step in our attack pipeline to validate the perturbations applied to a feature vector as shown in Figure 2.3. That is, attacks are initially applied to malware samples to generate adversarial examples without any limitations. Then, because the attacks may produce continuous feature vectors, their values are rounded to the nearest integer (i.e., 0 or 1) to represent the presence or absence of that feature. Each adversarial example is then inspected for prohibited perturbations, which are reversed. As this process can change back features used to cross the decision boundary, we then ensure that the adversarial example is still adversarial by testing it on the model.

3.4.3.2 Procedure

As detailed in Section 3.2, our threat model mainly consists of a gray-box scenario where the attacker's knowledge is limited [162, 118, 185, 28], so we focus in this section on describing the process we follow for generating

adversarial examples under this threat model. We also consider a black-box scenario, but this is described in detail in Section 3.5.5. In particular, for the gray-box scenario, attackers have access to the same training data as the target model and have knowledge of the feature representation. Therefore, to simulate this scenario, we construct four substitute vanilla models using the training data: a decision tree (DT), neural network (NN), random forest (RF) and support vector machine (SVM). In Table 3.2, we provide the architectures of these models (that is, we use the same model architectures for the vanilla models).

It is well-established that using models with architectures different from the target model (e.g., StratDef or some other model) can be used to evade it [207]. Therefore, we apply the attacks listed in Table 3.3 against these substitute models to generate adversarial examples. We can apply white-box attacks to the substitute models because we have access to their gradient information.

Attack	Applicable Model Families
Basic Iterative Method (A) [116]	NN
Basic Iterative Method (B) [116]	NN
Boundary Attack [36]	DT, NN, RF, SVM
Carlini Attack [45]	NN, SVM
Decision Tree Attack [89]	DT
Deepfool Attack [142]	NN, SVM
Fast Gradient Sign Method [87, 70]	NN, SVM
HopSkipJump Attack [50]	DT, NN, RF, SVM
Jacobian Saliency Map Approach [160]	NN, SVM
Project Gradient Descent [134]	NN, SVM
Support Vector Machine Attack [89]	SVM

Table 3.3: Attacks used to generate adversarial examples. Some attacks can only be applied to certain model families.

An overview of the procedure for generating adversarial examples is as follows:

1. With an input malware sample and an (applicable) substitute model (S), an attack (τ) is performed, to generate an adversarial example. The malware samples are those from our test set.
2. If the generated feature vector is continuous, the values within are rounded to the nearest integer (i.e., 0 or 1), in order to restore it to a discrete feature vector. For example, if an attack changes the value of a particular feature to < 0.5 , it is set to 0 in the feature vector; meanwhile, if the value is ≥ 0.5 , it is set to 1 in the feature vector.
3. We then verify that all perturbations used to generate the adversarial example are valid according to the dataset. Any invalid perturbations are reverted to offer a lower bound of functionality preservation within the feature-space, similar to prior work [192, 90, 124].
4. The adversarial example is then evaluated to ensure it is still adversarial. For this, the substitute model S makes a prediction for the original input sample and the adversarial example; a difference between the predictions indicates that the adversarial example has crossed the decision boundary. If so, it is added to the set of adversarial examples generated by attack τ against S , which is represented by $\Omega_{\tau, S}$.

The sets of adversarial examples can be used to evaluate StratDef and the other models, following the steps in the next sections. The process described above results in 4,608 unique adversarial examples for DREBIN and 5,640 for SLEIPNIR. The procedure for generating adversarial examples is performed by the defender and the attacker independently. Different attacker profiles are then constructed as described in Section 3.4.4 based on the generated adversarial examples. The defender uses the generated adversarial examples (together with the training and validation data) as part of the process described in the next section for constructing some of the models and defenses.

3.4.4 Modeling Gray-box Attacker Profiles

After generating each set of adversarial examples $\Omega_{\tau,S}$ as detailed in the previous section, we assign each set to different attacker profiles, according to Table 3.4. The aim of this is to simulate and evaluate StratDef’s performance against different types of attackers with varying capabilities, similar to prior work [44, 46, 28, 203, 192, 168].

When modeling attacker profiles, we ensure that the strongest attacker only uses the sets of adversarial examples with higher normalized evasion rates¹ against each model $F \in \Sigma_\gamma$. Weaker attackers use those sets with lower normalized evasion rates. Additionally, stronger attackers can *observe transferability*. If an attacker cannot observe transferability, then when assigning them a set of adversarial examples, we only consider the normalized evasion rate against the substitute model S , which is the original applicable substitute model, and not against models in Σ_γ , which could be higher due to transferability.

Attacker	Strength	Observe Transferability	Pooling Procedure
Weak	$R_{\tau,S,F} \leq 40$	✗	Random
Medium	$40 \leq R_{\tau,S,F} \leq 80$	✓	Random
Strong	$80 \leq R_{\tau,S,F}$	✓	Weighted based on $R_{\tau,S,F}$
Random	Any	✓	Random
Universal	UAPs only	✓	Random

Table 3.4: Profiles of different gray-box attackers who interact with StratDef. For details of the black-box attacker, see Section 3.5.5.

Once an attacker has been assigned sets of adversarial examples, the sets are aggregated into a single set so that each attacker has a selection of adversarial examples to model and represent their capability. Using these, we create datasets to represent different attack intensities, denoted by α , for each attacker. The attack intensity represents the proportion of adversarial examples in the dataset (i.e., adversarial queries made by attackers). Each dataset corresponds to a value of $\alpha \in [0, 1]$ with increments of 0.1. For example, at $\alpha = 0.1$, 10% of the dataset consists of adversarial examples. The remaining 90% consists of an equal number of benign and *non-adversarial* malware samples from the test set described in Section 3.4.1. This would therefore represent a system condition (or operating environment) with a lower attack or adversarial intensity as compared with $\alpha = 0.9$, for example.

¹Recall that the normalized evasion rate encompasses the number of adversarial examples in a particular set and their overall evasiveness (see Section 3.3.3).

The *pooling procedure* in Table 3.4 governs how the adversarial examples for the α datasets are chosen. For the strong attacker, the construction of these datasets gives preference to more evasive adversarial examples from their aggregated set. That is, to construct the α datasets, adversarial examples that are from more evasive sets have a higher chance of being chosen. This is unlike for other attackers, where the adversarial examples are chosen randomly from their aggregated set. Meanwhile, the universal attacker only gets assigned adversarial examples with universal adversarial perturbations (UAPs) [143, 117]. In total, there are 1,541 such universal examples for DREBIN (UAP_{DREBIN}) and 2,217 for SLEIPNIR ($UAP_{SLEIPNIR}$) (see Section 3.5.4 later).

The datasets generated for each attacker are also used by the defender for developing strategies (as per Section 3.3.2) and evaluating the performance of each defense by simulating attackers with different levels of adversarial queries. We next present our experimental evaluation and results.

3.5 Evaluation & Results

In this section, we present an evaluation across Android and Windows in the experimental setting described previously. In Section 3.5.1, we analyze the performance of StratDef under different threat levels. Then, we compare StratDef’s performance with other defenses in Section 3.5.2 and compare its efficiency with other defenses in Section 3.5.3 later. Following this, we evaluate how StratDef performs against Universal Adversarial Perturbations in Section 3.5.4. Finally, we show how StratDef copes with a complete black-box attacker in Section 3.5.5.

3.5.1 Performance of StratDef

In this section, we present the results for the StratDef configurations against different attackers and attack intensities. In our discussion, we focus on the aggregate results but refer the reader to Appendix B.2 for extended results.

Figure 3.3 demonstrates that as the threat level increases (with stronger attackers and higher intensities), there is a greater effect on the performance of StratDef. Notably, as the threat level increases, we observe decreases in accuracy in many cases, as one may expect. However, at the peak threat level, StratDef can achieve up to 52.6% accuracy for DREBIN and 100% accuracy for SLEIPNIR, with the highest average accuracy of 72.7% for DREBIN and 96.2% for SLEIPNIR across all attack intensities for all configurations. These results demonstrate that the adversarial examples for DREBIN are more evasive than those for SLEIPNIR, as indicated by StratDef’s lower accuracy. This is likely because there are more perturbations that can be used to generate adversarial examples with DREBIN (i.e., feature addition *and* removal). For SLEIPNIR, the weakest StratDef configuration (StratDef-Best-URS) only drops to $\approx 80\%$ accuracy at the peak threat level. The weaker adversarial examples for SLEIPNIR can be attributed to more limitations in the perturbations that can be applied, therefore reducing the attack surface.

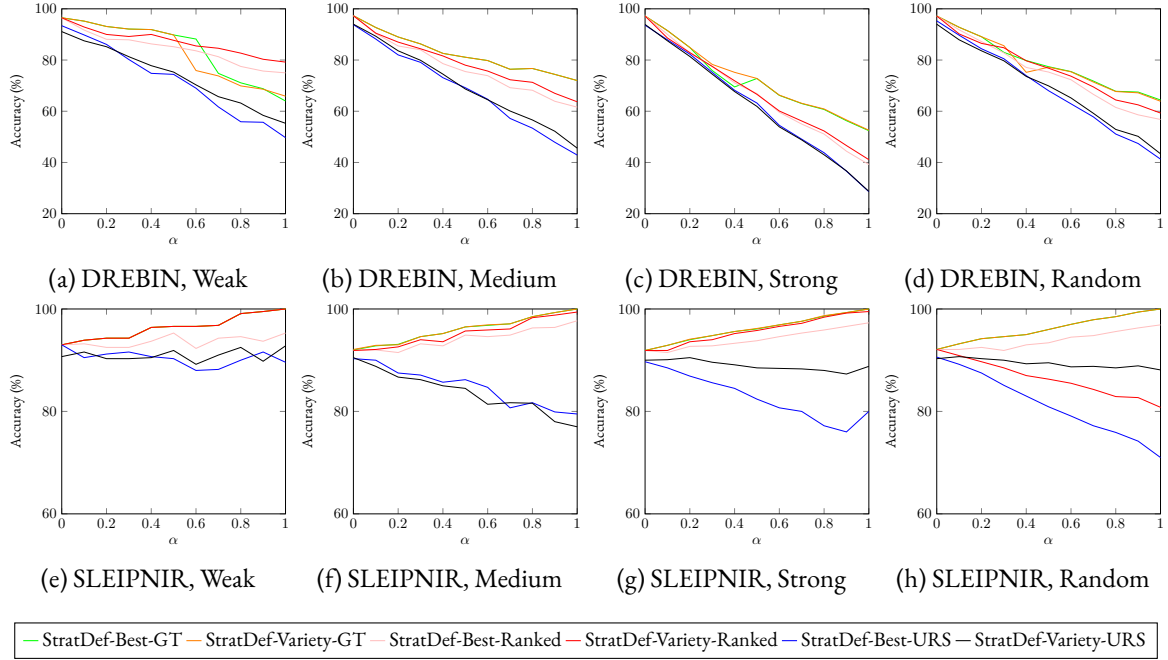


Figure 3.3: Accuracy of different StratDef configurations against different attackers under varying attack intensities. In Figures 3.3b, 3.3e and 3.3h, StratDef-Best-GT and StratDef-Variety-GT have the same performance because of identical strategies.

In terms of the model selection, the configurations using the *Variety* model selection perform well at all threat levels. Recall that this model selection enforces diversity among the constituent models of the ensemble (see Section 3.3.2). The results show that the increased diversity in this model selection offsets the transferability of adversarial examples better than the configurations using the *Best* model selection. This is further reflected, albeit marginally, across all metrics, as visible in Figure 3.4. This figure shows the values of different metrics for StratDef configurations averaged across attack intensities for both datasets against the strong attacker.

Meanwhile, regarding the optimizer, the *game-theoretic* (GT) configurations offer slightly better performance for both datasets than the configurations using the *strategic ranked optimizer* (Ranked). However, a potential limitation of the GT configurations is that they switch between pure and mixed strategies, with adversarially-trained models featuring more often in the strategies. In fact, with the GT configurations, only up to 30% of the model selection is used against the strong attacker, meaning that the majority of the model selection may never actually be used. It is important to understand that employing a StratDef configuration (or an MTD) that uses fewer models may increase the risk of an attacker discovering the profile and configuration of the deployed defense. However, due to transferability, using more models may open an avenue for greater evasion as there is a higher chance that a vulnerable constituent model is chosen at prediction-time. Therefore, an inherent trade-off exists between the number of models used in the ensemble and the robustness of the system. If a more diverse set of models is used to reduce the transferability, the attacker will be less successful.

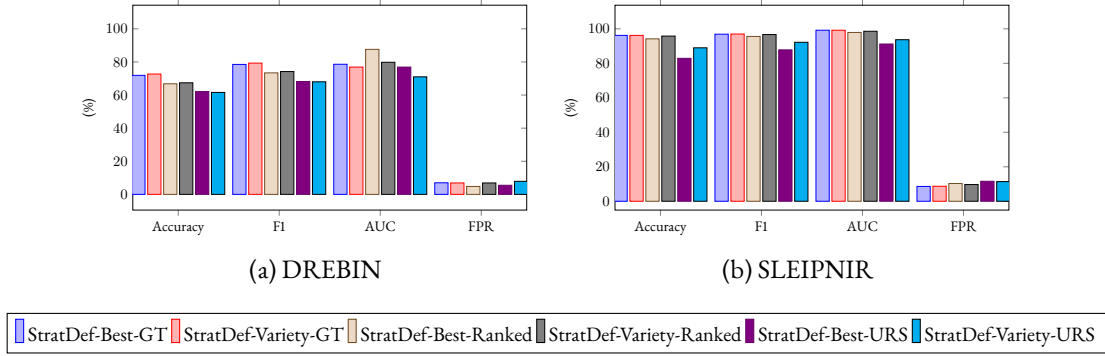


Figure 3.4: Average values of metrics across attack intensities for StratDef configurations against the strong attacker.

The configurations using the strategic ranked optimizer only produce mixed strategies since this optimizer does not give complete preference to the strongest model. Intriguingly, despite using more models, these configurations offer similar performance to those that use the GT optimizer. This is especially visible in Figure 3.4, where we observe that the F1 and AUC tend to remain generally high, while the FPR tends to remain quite low, for the GT and Ranked configurations across the board. Overall, these results illustrate the capability of our strategic defense to produce correct predictions across a number of scenarios, especially adversarial ones, which is quite encouraging. In contrast, the poorer performance expected of the uniform random strategy (URS) approach can be seen in Figures 3.3 and 3.4, which highlights the need for good movement strategies regardless of how sound the model selection itself is. Recall that under this configuration — which is only included as a baseline — a model is chosen randomly from the ensemble to serve a user’s prediction, rather than with a strategic or heuristic approach. The results show that using a randomized approach for choosing models at prediction-time is no match for a game-theoretic or heuristically-driven strategy. For maximal robustness, only strategic or game-theoretic movement approaches are suitable.

3.5.2 StratDef vs. Other Defenses

We next compare StratDef with a range of other models and defenses that we introduced in Sections 2.3.1 and 3.4.2.1. We present a selection of the results in Figure 3.5 covering a number of defenses that we evaluate. In our discussion, we focus on the aggregate results specifically under the strong attacker scenario. For extended results, we refer the reader to Appendix B.2.

Overall, our evaluation shows that the other models and defenses that we evaluate perform significantly worse than StratDef, except for high levels of adversarial training (which we discuss further later). This is especially clear at the highest threat level, with the highest attack intensity and the strongest attacker. For example, as one might expect, the vanilla NN and SVM models perform poorly and only achieve 7% and 2% accuracy, respectively, at the peak threat level for DREBIN, with only marginally better performance for SLEIPNIR. However, even after some robustness procedures are applied to these models, the performance only improves by $\approx 10\text{-}15\%$ in accuracy for defensive distillation (NN-DD) and SecSVM. The random feature nullification (RFN) approach offers an insignificant improvement to the vanilla models from all model families,

especially, at the peak threat level. These results demonstrate the general ineffectiveness of these defensive approaches. Interestingly, in some instances, the vanilla random forest (RF) for DREBIN and the decision tree (DT) for SLEIPNIR can even surpass defenses such as random feature nullification (RFN), though for SLEIPNIR at the cost of a higher number of false positives. Nonetheless, all of these defenses and models still significantly underperform in comparison to our StratDef approach as shown in Figure 3.5, whose average accuracy across the attack intensities sits comfortably above 70% for both datasets.

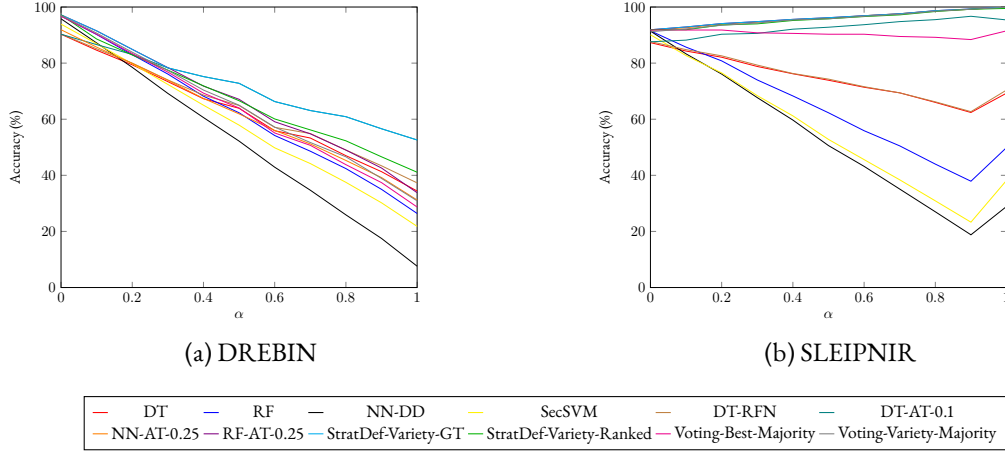


Figure 3.5: Comparison of StratDef with a range of other defenses against the strong attacker.

StratDef also outperforms the voting defense. At the peak adversarial threat, the Voting-Best-Majority and Voting-Variety-Majority configurations are on par with the vanilla models and other defenses for DREBIN, only achieving a maximum of 30% accuracy. For SLEIPNIR, although the voting defense can achieve 90+% accuracy with adequate F1, this comes at the cost of slightly higher FPR and FNR as well as a lower AUC compared with the StratDef Best and Variety configurations (see Appendix B.2).

Generally, defenses for SLEIPNIR are more consistent and appear to work better because of fewer allowed perturbations, reducing the attack surface and limiting the avenues for evasion. Meanwhile, in the more complex scenario with DREBIN, the attacker has greater opportunity with the perturbations. There, our evaluation shows that StratDef is much superior in dealing with adversarial examples across the attacker scenarios.

Intriguingly, from the other defenses we evaluate, we find that only high levels of adversarial training provide any notable robustness. This is similar to findings in previous work [169, 214, 191]. For example, at the peak threat level, 50+% accuracy for DREBIN and 90+% accuracy for SLEIPNIR can be achieved with high levels of adversarial training across the model families (typically the 0.1-0.25 variants). As a side-effect, however, the adversarially-trained models can sometimes result in more false positives. For example, the highly adversarially-trained models based on the NN and SVM model families for DREBIN, and DT for SLEIPNIR cause more false positive predictions, though this is less observable with the adversarially-trained RF model in both datasets.

Overall, we observe that highly adversarially-trained RF and DT models are the most balanced all-rounders from existing defenses for both datasets, offering decent F1 and AUC while maintaining fewer false predictions than other defenses. In some cases, while StratDef may perform similarly to such adversarially-trained models,

a significant advantage of StratDef is that it simplifies the process of selecting an appropriate model to deploy. Additionally, StratDef has a benefit over single adversarially-trained models as it complicates the attacker’s efforts to construct substitute models, reducing the success of black-box attacks as it does not behave like a fixed, static target (see Section 3.5.5 and even Chapter 4 later).

3.5.3 Efficiency of StratDef

We also assess the prediction-time efficiency of StratDef, voting, and some of the other best-performing defenses. This is important, as we hypothesize that defenses may differ in their prediction times and resource consumption due to their particular configuration. In this section, we focus on the prediction-time efficiency allowing us to assess the quality of service for users as if StratDef were to be deployed. Since model training is not a core part of the StratDef framework — which specializes in selecting the best models and how to strategically use them — we do not focus on the training-time efficiency in this work. Recall that StratDef can also consider pre-trained models for other work, and with enough resources, a defender could minimize any training-time costs.

For this experiment, we query each considered defense 1000+ times continuously and then measure the time taken to produce a prediction. Figure 3.6 shows the average time taken by each defense to produce a single prediction against the strong attacker. Evidently, there is a significant time difference between StratDef and voting, with voting exhibiting higher costs. This is because, with voting, all constituent models must be waited for while they produce their individual predictions before the final prediction can be returned. Meanwhile, StratDef returns predictions in a similar time to single-model defenses, as only a single constituent model is used for a prediction, with minimal overhead involved in rolling a biased die to choose that particular model at prediction-time. In fact, StratDef returns faster predictions on average than RF-AT-0.25 for both datasets. Intriguingly, some prior work has found that random forests are generally slower than other models to produce predictions [38, 40, 210], and in our work, this also seems to be the case in this experimental setting when comparing random forests with StratDef.

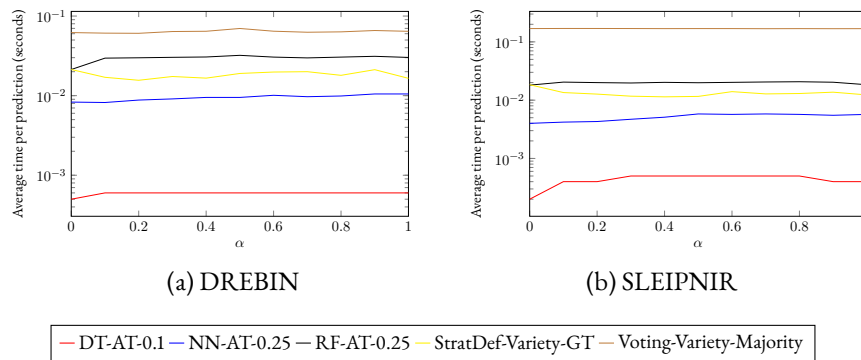


Figure 3.6: Average time taken per prediction against the strong attacker.

We also examine the average memory consumption incurred when producing a prediction against the strong attacker. This is because we are interested in analyzing the overhead of keeping models in memory to serve predictions. Figure 3.7 shows the average memory consumption of each defense across attack intensities.

Defenses besides StratDef have static memory consumption because they are not strategized for the attack intensity. The single-model defenses we evaluate consume less memory, with minor differences due to the particular model family. Meanwhile, ensemble defenses require access to more models at prediction-time, leading to higher memory costs. However, voting has the highest memory consumption since it uses all models in the ensemble for a single prediction. For example, in the case of Voting-Variety-Majority for SLEIPNIR, a single, memory-intensive model requires over 0.7GB memory. Meanwhile, StratDef is efficient and better than voting, as it only loads models for each attack intensity with a non-zero probability (i.e., those that have a non-zero chance of being chosen to make a prediction) rather than keeping all constituent models in memory. Overall, StratDef — which is an ensemble defense — performs as efficiently as (and sometimes better than) single-model defenses, considering both time and memory costs.

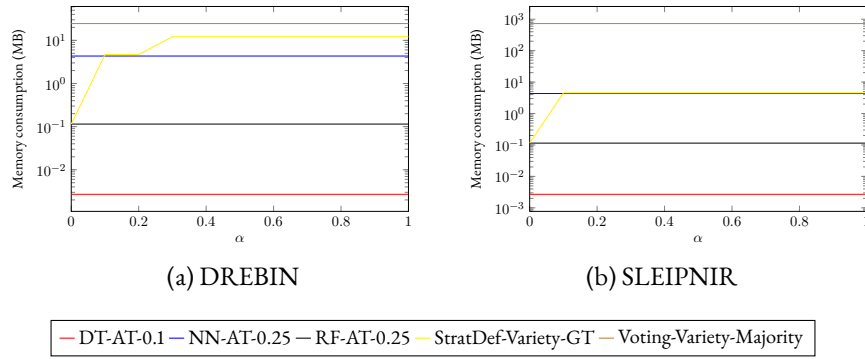


Figure 3.7: Average memory consumption of models against the strong attacker.

3.5.4 StratDef vs. UAPs

Recent work has uncovered universal adversarial perturbations (UAPs) as a cost-effective method of generating adversarial examples where perturbations can be reused [143, 117] (see Section 2.2.5.2 for a background discussion on UAPs). We therefore evaluate how StratDef performs against the *universal attacker*, who only uses adversarial examples generated from UAPs. For this, we determine if a set of perturbations has been precisely reused to generate adversarial examples (from Section 3.4.3). In such situations, adversarial examples are regarded as having been generated by UAPs. For example, if the application of the same set of perturbations to several distinct malware samples has produced adversarial examples, the set of perturbations counts as UAPs. In total, there are 1,541 such adversarial examples for DREBIN (UAP_{DREBIN}) and 2,217 for SLEIPNIR ($UAP_{SLEIPNIR}$). We present the results of this experiment in Figure 3.8 (with Appendix B.2 containing all results).

Figure 3.8 shows the accuracy of the StratDef configurations and other best-performing models against the universal attacker. The adversarial examples in UAP_{DREBIN} and $UAP_{SLEIPNIR}$ appear less evasive than before, as the accuracy of most models is hardly affected. That is, the universal attacker is unable to achieve effective evasion against most defenses, including any StratDef configuration, with its accuracy remaining largely over 90%. Only vanilla and weakly-defended NN and SVM models fall prey to the universal attacker, with their accuracy reduced to less than $\approx 40\%$ across both datasets (these are not shown in Figure 3.8 but are

available in Appendix B.2). Evidently, the results are more closely concentrated for StratDef, with generally sound performance across both datasets, particularly in its GT configurations.

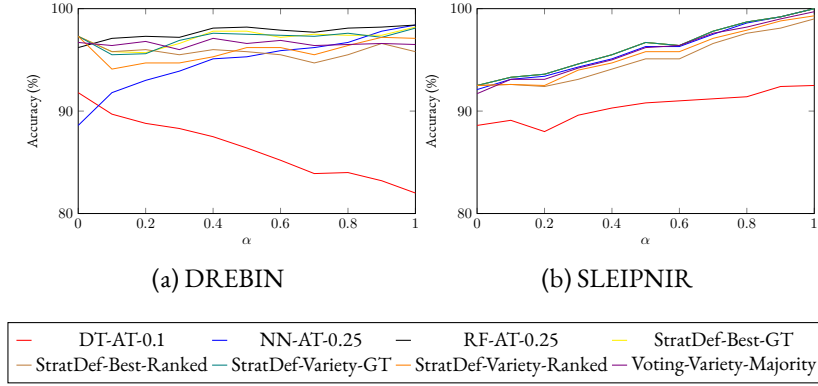


Figure 3.8: Accuracy of different defenses against the universal attacker. For SLEIPNIR, StratDef-Best-GT and StratDef-Variety-GT have the same performance. Some models have similar performance (see Appendix B.2 for extended results).

3.5.5 StratDef vs. Black-box Attacks

In this section, we explore how StratDef performs against a black-box attacker. Under this attack scenario, a zero-knowledge attacker — who can only observe the predicted outputs of the target model — sends queries to the target model as if it were an *oracle* about which nothing is known. The black-box attacker uses the input-output relations to develop a substitute model (i.e., an estimate of the target model) that they can attack [157]. The attack strategy employed here is an example of a black-box transferability attack strategy (see Section 2.2.5.2). Later, in Chapter 4, we provide a more detailed experimental evaluation of different black-box transferability attack strategies against MTDs.

3.5.5.1 Procedure

For this experiment, we follow the standard procedure to attack a target model using a black-box transferability attack strategy as described in Section 2.2.5.2. This is based on the well-established idea that adversarial examples generated for a substitute model may transfer to the target model [207].

Initially, we query the target model (whether it is StratDef or some other defense) with an equal number of benign and malware samples² and record the predicted outputs from the model. We vary the *number of [these] samples at training-time* to examine if this affects the success of attacks, as more interactions at training-time should produce a better representation of the target model. The input-output relations from querying the target model are then used to train a substitute DNN (see Table 3.2 for the model architecture), which acts as an estimation of the target model. Against this substitute DNN, we then use white-box attacks (BIM [116], FGSM [87, 70], JSMA [160] and PGD [134]) to generate adversarial examples, in anticipation that they will transfer

²Therefore, Figure 3.9 starts with 2 samples because a single sample from each class is used to build the training set for the substitute DNN.

to the target model. As before, procedures are applied to ensure a lower bound of functionality preservation in the feature-space when generating the adversarial examples (see Section 3.4.3). The adversarial examples are then tested against the target models, such as StratDef and other defenses.

In this attack setting, StratDef assumes the highest threat level (i.e., the strongest attacker at the highest attack intensity). Beyond this, StratDef is not strategized to deal with a black-box attack of any kind. Therefore, this attack also helps us understand how StratDef may work against an unknown attacker.

3.5.5.2 Results

In Figure 3.9, we present the results of this experiment, where we compare StratDef with other well-performing defenses from the prior subsections. We find that StratDef works well across both datasets, with a relatively low evasion rate compared with other defenses. For DREBIN, the attacker achieves a 19% evasion rate against StratDef in the worst-case which is lower, and hence better, than other defenses, and around 16% evasion rate on average across all results, which is still lower than the other defenses. Recall that StratDef is not currently strategized to deal with such an attack despite its better performance. As StratDef cycles between models during predictions, we also observe variations in the attacker’s performance. Meanwhile, although the adversarially-trained DT model performs adequately against DREBIN, it performs much worse for SLEIPNIR, while other models also exhibit mixed performance. Regardless, in the more complex scenario involving DREBIN, StratDef offers superior performance against black-box attacks. There, we generally observe that as the number of samples at training-time increases, the evasion rate also increases up to a certain level. This supports the hypothesis that substitute models that are trained using a higher number of input-output relations of the target model are better representations of it.

For SLEIPNIR, the attacker is less successful, which is a theme we have seen previously. This is due to a more limited feature-space (i.e., the set of allowed perturbations for generating adversarial examples) and is reflected in the results for the black-box attack, where the evasion rate decreases considerably against the stronger defenses such as StratDef ($< 1\%$ evasion rate). However, the DT-based model is greatly evaded, with an evasion rate of 25+%. StratDef makes use of the DT-AT-0.1 model in its strategy at the considered threat level and therefore suffers slightly in comparison to other models.

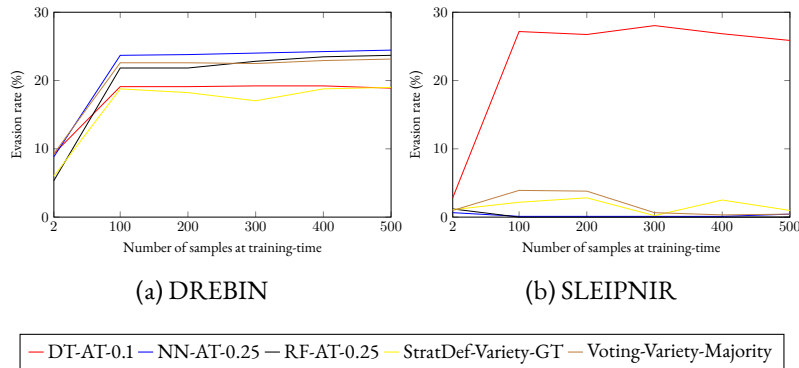


Figure 3.9: Results of black-box attack against various defenses.

Interestingly, we also observe that the attacker’s success against some defenses fluctuates (e.g., after 200 samples for training). This is likely because the substitute model becomes noisier, especially in the case of voting. As voting uses multiple models for a prediction, there may be some predictions that are returned without a large majority for either class. This means that the training data for the substitute model becomes inaccurate, leading to poorer attack performance. Regardless, the results show that StratDef can offer a greater degree of protection against this type of attack than other defenses. We later expand on this attack strategy in Chapter 4.

3.6 Conclusion

In this chapter, we proposed StratDef, a strategic moving target defense against adversarial attacks in ML-based malware detection. StratDef embraces the key design principles of an MTD and provides a complete framework for building a strategic ensemble defense using different heuristically-driven methods for determining what, how, and when an MTD should move to achieve a high degree of adversarial robustness. In the experimental evaluation, we demonstrated the superiority of StratDef over other defenses across both Android and Windows malware under different attacker scenarios. We found the majority of prior defenses — bar adversarial training — to be insufficient at defending against adversarial attacks. Overall, we demonstrated the ability to construct a strategic MTD that can increase accuracy by 50+%, while reducing the success of targeted attacks by raising the uncertainty and complexity for the attacker. Although adversarial training performed well, we identified some issues with this method. For example, adversarial training introduces additional complexities such as determining which model to choose as the base model, what to train on, and how much to train. However, as we showed, StratDef assists with this, as it could help select the most promising models and then choose between them strategically at prediction-time. As we showed in the evaluation, this produces better results across a range of attacker scenarios than a single model trained adversarially.

The results and conclusions of this chapter further motivate the idea that a strategic MTD against adversarial attacks can increase robustness better than other types of defenses. However, in recent years, several other MTDs for adversarial ML have also been proposed. These MTDs have not been widely evaluated, nor compared with each other, or other types of defenses extensively, nor characterized. Hence, it is imperative to assess whether MTDs, as a whole, are a line worth investigating further, and, if so, what kinds of strategies and ways to design MTDs are more effective. Therefore, in the next chapter, we conduct the first comprehensive evaluation of several recent MTDs in order to characterize their performance and derive the best practices for developing robust MTDs against adversarial ML. Based on the empirical study, we provide several key recommendations for building MTDs in future work, underpinned by our investigation of recent MTDs.

CHAPTER 4

EFFECTIVENESS OF MOVING TARGET DEFENSES AGAINST ADVERSARIAL ATTACKS

In this chapter, we study the effectiveness of several recent moving target defenses for adversarial ML applied to ML-based malware detection. Previously, we presented StratDef, which is a defense based on a strategized moving target defense (MTD) approach. We demonstrated its ability to outperform other defenses even against the peak adversarial threat in this domain. In recent years, a number of other MTDs for adversarial ML have also been proposed. As discussed in Section 2.4.1, each of these MTDs has introduced its own movement methods and strategies in order to try and increase the difficulty for the attacker in conducting attacks. However, despite several studies examining the effectiveness of other types of defenses for adversarial ML, the performance of MTDs under different threat models has not been evaluated, nor compared with each other, nor extensively compared with other defenses.

It is imperative to assess whether MTDs, as a whole, may be a line worth investigating further to remedy adversarial ML attacks, and if so, what kinds of strategies and ways to design MTDs seem more effective. Therefore, in this chapter, for the first time, we study, characterize, and evaluate moving target defenses for adversarial ML in ML-based malware detection. Against a range of attacks, we discover that high levels of evasion against most MTDs can be achieved even in settings where the attacker’s knowledge is restricted. Moreover, we discover that fingerprinting and reconnaissance activities — which MTDs specifically aim to make more difficult — may be possible with a set of introductory attacks. Crucially, based on our findings, we present key recommendations for spurring future work on the development of effective MTDs for adversarial attacks in ML-based malware detection. The work in this chapter relates to OBJ3, where we aim to evaluate and characterize several MTDs for adversarial ML.

4.1 Introduction

Recently, numerous MTDs have been proposed as a remedy against adversarial ML attacks (e.g., [174, 191, 171, 13, 200, 58]). As discussed in Chapter 2, MTDs use several ML models (rather than a single model) in a specific manner to defend against adversarial attacks. The overall aim of an MTD approach is to increase the complexity of conducting reconnaissance and targeted attacks by regularly moving aspects of the defense’s configuration (e.g., cycling through constituent models or how predictions are produced) [58]. A number of techniques to construct MTDs for adversarial ML have been proposed, such as increasing the heterogeneity

of constituent models, diversity training of constituent models, dynamically regenerating the models, query-limiting, and strategically choosing models (see Sections 2.3.2 and 2.4.1 for detailed discussions on different MTD approaches). While prior work has shown the ineffectiveness of single-model defenses for adversarial ML (see Sections 2.4.1 and 2.4.2), these MTD approaches have not been widely evaluated, nor compared with each other, nor extensively compared with other defenses. That is, the performance and behavior of MTDs as a whole have not been characterized through experimental work prior, to the best of our knowledge.

Hence, in this chapter, we present the first evaluation of several recent MTDs where we compare their performance with each other and other types of defenses under different threat models for the first time using a range of attacks. This chapter is a comprehensive evaluation of several MTDs as a family of defenses for adversarial ML, and not just an evaluation of StratDef (which we proposed in the previous chapter).

To conduct the evaluation, we use transferability and query attack strategies from prior work, as well as novel ones that we propose, in order to maximize the evasion of MTDs. Worryingly, the results suggest that even in the most restricted attacker scenarios, it is still possible to achieve a 90+% evasion rate against MTDs. Additionally, in this work, we introduce methods to conduct fingerprinting and reconnaissance in order to increase the attacker's understanding of how a target MTD works for enhancing future attacks, with minimal knowledge about it initially. This means that attackers may obtain valuable information such as critical defense hyperparameters as well as knowledge about how predictions are produced.

Informed by the evidence produced in our evaluation, we derive key insights and make crucial recommendations to help design more effective MTDs against adversarial ML attacks in the ML-based malware detection domain. In particular, we recommend that MTDs that utilize diverse constituent models with effective movement should be favored, while MTDs that exhibit static behavior should be avoided. That is, in order to maximize diversity for limiting the effectiveness of transferability attacks, an MTD should use constituent models from different families. Importantly, we recommend in this chapter that these constituent models need to be used dynamically and moved strategically to maximize uncertainty and complexity for the attacker. Crucially, we also recommend that MTDs be coupled with greater system awareness to detect and react to *attacks in progress*. This would pave the way for a promising way forward in dealing with query attacks.

Next, we present the threat model and attacker scenarios used in this chapter to evaluate the effectiveness of MTDs.

4.2 Threat Model

We next describe the threat model used in this chapter. Like in the previous chapter, we model the attackers' primary aim as evading a feature-based ML model in order to have malware samples predicted as benign by that target model.

4.2.1 Target Model

The target model is an MTD that consists of several constituent ML models that are used in a particular manner to produce predictions. We refer the reader to Section 2.3.2 (and notably Figure 2.7) for a thorough discussion of MTDs for adversarial ML that employ multiple constituent models.

Recall that to train the constituent models, software executables are represented as binary feature vectors based on raw extracted features. These constituent models are binary classification models for ML-based malware detection (like the application model described in Figure 2.1). Following established practices in ML-based malware detection, the features we use include libraries, API calls, permissions, and network addresses accessed, among others, which are provided by the datasets we use, detailed later in Section 4.4. As before, with the features $1 \dots M$, a vector X can be constructed for each input sample such that $X \in \{0, 1\}^M$. Here, $X_i = 1$ or $X_i = 0$ indicates the presence or absence of feature i respectively. The feature vectors and their associated class labels are used to train the constituent binary classification models, as depicted in Figure 2.1.

The constituent models are then used in an MTD known as the *oracle* O . A user requests a prediction from O , which uses the constituent models in a specific manner to return a prediction (e.g., a probability distribution to *move* between, i.e., choose, the constituent models [191]). The oracle provides scoreless feedback [102], only returning the prediction to limit the user’s understanding of its behavior.

4.2.2 Attacker’s Goal

As described in the previous chapters, the attacker’s goal is to generate an adversarial example X' to evade the target model, O . Suppose $O: X \in \{0, 1\}^M$ and we have some function *checkFunctionality*() to check the functionality of an input sample. This goal can then be summarized as:

$$\text{checkFunctionality}(X) = \text{checkFunctionality}(X'); O(X) = 1; O(X') = 0 \quad (4.1)$$

That is, X' must be functionally-equivalent to the original sample X but result in a *benign* prediction from O . For the predicted outputs, recall that 1 represents the malware class and 0 represents the benign class.

4.2.3 Attacker Capabilities & Knowledge

We model two types of attackers with different levels of knowledge, as commonly featured in prior work [102, 118, 162]. Importantly, neither attacker knows that the target model is an MTD.

The limited-knowledge *gray-box attacker* has access to the same training data as the target model and has knowledge of the feature representation as well as the statistical representation of the features across the dataset. However, they have no knowledge of the parameters, configurations, or constituent models of the target model. This could represent an example of when some sensitive model information may have been leaked. Therefore, following the discussion in Section 2.2.5.2, which we expand on later, for transferability attacks, the gray-box attacker trains substitute models using the training data and attacks them with the aim of having the adversarial examples transfer to the oracle [207, 118, 157]. Meanwhile, to conduct query attacks, the gray-box attacker

uses their extensive knowledge to apply suitable perturbations using a software transplantation-based approach in a heuristically-driven manner.

Meanwhile, the *black-box attacker* can only observe the predicted outputs for their queries. They have no knowledge about the target system but have some information pertaining to the kind of feature extraction performed (e.g., the static analysis that a malware detection classifier may consider). Therefore, they conduct a chosen-plaintext attack [48] for transferability attacks. As we describe in the next section, this is achieved by querying the oracle sufficiently to train a substitute model, which is an estimation of the oracle, and then attacking it to generate adversarial examples that transfer to the oracle [207, 118, 157]. This is unlike the gray-box attacker, who can use the training data to train substitute models. For query attacks, the black-box attacker also uses a transplantation-based approach but without any extra information to guide the attack [182, 168].

Having detailed the threat model, we next describe the attack transferability and query attack strategies used in this chapter for evaluating the MTDs.

4.3 Attack Strategies

In this section, we present the attack strategies that are used to evaluate MTDs, considering the threat model described in the previous section. We use a combination of transferability and query attack strategies from prior work and also propose strategies with novel improvements to these for maximizing the evasion of MTDs.

4.3.1 Transferability Attacks

In transferability attacks, attackers construct substitute models that are approximations of the oracle, as detailed in Section 2.2.5.2. Adversarial examples are then generated for these substitute models in anticipation that they will transfer to the oracle [87, 157, 160, 207, 181, 36, 131]. To evaluate MTDs against transferability attacks, we use practical strategies from prior work as well as our own novel ones. Attack strategies from prior work are included as baselines and to show MTDs' performance with already existing, general strategies that are not tailored to MTDs. These include the strategies discussed in Section 2.4.2; namely the Single DNN strategy [157], where a single DNN is used as the substitute model [157, 181, 36], and the Ensemble DNN strategy [131], where several DNNs are used as the substitute models.

Additionally, we propose a novel attack strategy that specifically considers that the target model *may be* an MTD. Even if, per our threat model, attackers do not know the target model is an MTD, we can try this attack strategy to achieve maximal evasion against such target models, nonetheless. With our transferability attack strategy, we present two main improvements over previous strategies, with an overview of the key steps provided in Figure 4.1. First, our strategy utilizes an ensemble of diverse substitute models (including different ML families). Second, as a novel technique to increase attack success against MTDs, our attack strategy aims to maximize the transferability of adversarial examples across substitute models prior to evaluating them on the oracle by checking transferability across (part of) the substitute models. We include these two additions because MTDs may change the model used for predictions dynamically, and so ensuring a degree of transferability of

adversarial examples between diverse substitute models maximizes success on the oracle, as we confirm later on experimentally. We next describe each of the key steps of our strategy in detail.

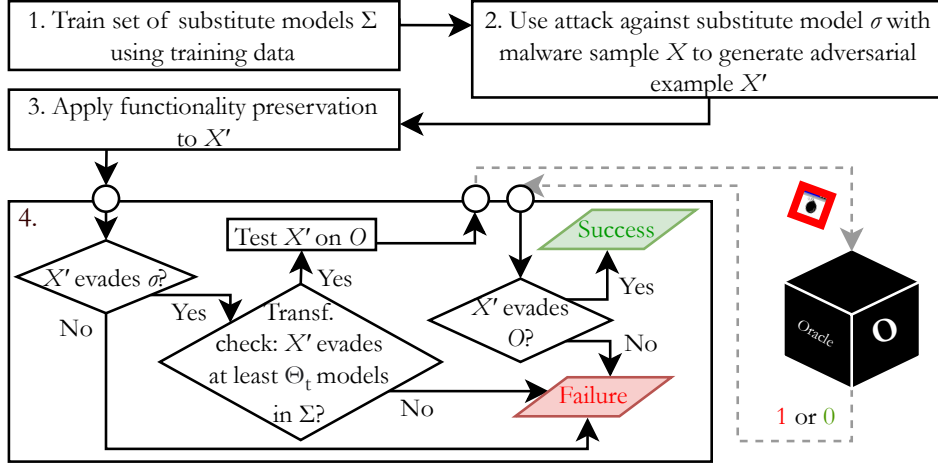


Figure 4.1: Overview of our transferability attack strategy.

Step 1) Developing Substitute Models

The aim here is to train substitute models that are an estimation of O (the oracle). It is well-established that a model can be evaded by utilizing different model architectures and a dataset independent of it [207]. As detailed in Section 4.4.5, we particularly favor an ensemble of substitute models as diverse as possible to increase the chances of finding highly-transferable adversarial examples that will transfer to the oracle.

For each attacker, the procedure for constructing the substitute models differs due to their capabilities. The *black-box attacker* has no capabilities beyond observing the predictions for their queries. Therefore, under this scenario, we develop a synthetic dataset (Δ) that is an estimation of the input-output relations of the oracle. To develop Δ , we use a set B_{train} (containing benign and malware input samples) and query O with each input sample $X \in B_{train}$, recording the predicted output. For example, if $O(X) = 0$, then the input-output relation $X \mapsto 0$ is stored in Δ . Δ is then used to train the ensemble of diverse substitute models (Σ) for our attack strategy. This provides white-box access to models that are an estimation of the oracle. Prior work in other domains has suggested that substitute models require a large training dataset [157] and, while impractical, an attacker could create an exact replica of the oracle with close to infinite queries. However, we show later that a high evasion rate can be achieved with a small set of input samples (i.e., $|B_{train}| \leq 100$). With fewer queries to the oracle, an attacker can remain stealthier, with lower chances of their malicious behavior being detected [52, 89].

Note that for the other attack strategies that we test (the Single DNN [157] and Ensemble DNN [131] strategies), Δ is used to train the substitute model(s) according to their procedures.

In contrast to the black-box attacker scenario, the *gray-box attacker* has access to the training data of the defense. Therefore, we train an ensemble of diverse substitute models (Σ) using this training data. The

advantage is that no queries to the oracle are necessary, reducing the time cost and the risk of adversarial behavior being detected. However, as we demonstrate later in Section 4.6, using the training data of the target models may not always lead to the best attack performance. Because they are based on direct queries to each oracle, the substitute models developed by the black-box attacker may reflect their characteristics and behavior better. An alternative technique, given that the gray-box attacker has knowledge of the dataset, is to use universal adversarial perturbations (UAPs) [143, 117]. With UAPs, the same sets of perturbations are reused and applied to several malware samples to generate adversarial examples. However, we show later in Section 4.6.3 that this attack approach is less successful.

Step 2) Generating an Adversarial Example

As usual in all transferability attacks, once the substitute models have been constructed, we have full access to them. Then, we can use an attack against a substitute model such as existing white-box attacks from the literature to generate an adversarial example X' from a malware sample X (e.g., FGSM [87], JSMA [160] — see Section 4.4.6 for the full list of white-box attacks used in the evaluation). The idea is that the adversarial example X' is likely to *transfer* to the target model (the oracle).

Step 3) Applying Functionality Preservation

We then validate the perturbations that have been used to generate X' . This is done in order to preserve the original malicious functionality of X within the feature-space as a lower bound, which is essential in this domain. Otherwise, one may have an adversarial example that evades the oracle but has lost its malicious functionality. In this process, any invalid perturbations found in X' are reverted to their original values from X . Note that the perturbations that are valid or invalid and the way in which functionality preservation works in practice depend on the particular target platform for the malware. We detail valid and invalid perturbations for each of the cases we explore in the evaluation (for Android and Windows) later in Section 4.4.6.

Step 4) Transferability Across Substitutes & Evaluation

We then ensure that the adversarial example X' still evades the substitute model used to build it, as the process for validating the perturbations may have reversed some perturbations used to cross the decision boundary. If so, after this, we include a crucial new step where we look at how successful X' is at evading the other substitute models. That is, a local transferability check is performed before actually testing X' on the oracle O . As the attack success relies on the transferability property holding between Σ and O , our hypothesis is that adversarial examples that transfer better across the ensemble of substitute models Σ are more likely to evade O . For this, we check whether X' transfers across a proportion of substitute models (Θ_t) before testing it on O . For example, $\Theta_t = 0.75$ means that X' must evade 75% of substitute models. If X' adequately transfers across the substitute models and then evades O , it is counted as a success, whereas a failure is when O is not evaded by X' .

4.3.2 Query Attacks

As discussed in Section 2.2.5.2, query attacks generate adversarial examples by iteratively perturbing an input sample [52, 36, 50, 182, 102, 129, 26], rather than using substitute models. Most query attacks, however, are designed for the image recognition domain and therefore perturb features continuously, meaning they are less effective in our domain due to its constraints, such as discrete features and functionality preservation. For example, as explained in Section 2.2.3, a feature for an API call (e.g., `CreateFile()`) cannot be perturbed continuously (e.g., `CreateFile() + 0.05`). For this, an entirely new feature is required, offering the same functionality.

To overcome these challenges, we can use software transplantation-based approaches as presented in Figure 2.5. This means that features from benign samples are used to perturb a malware sample (e.g., a feature is added to a malware sample), which can be conducted in a scenario with less [182, 229] or more [168] information about the target model. Overall, this allows malware samples to cross the decision boundary of the oracle while catering to the constraints of this domain. When conducting this attack, limiting the number of queries to the oracle is critical, as adversarial behavior can be detected when analyzing queries for abnormalities [52]. Moreover, some MTDs use query budgets, which may hinder the construction of adversarial examples. Hence, we use the parameter n_{max} to govern the maximum number of allowed queries during an attack instance. We offer two approaches for performing query attacks in black-box and gray-box scenarios.

4.3.2.1 Black-box Query Attack

Under the black-box scenario, the attacker has no knowledge of the target model besides the predicted output. This means that the attack is conducted in a non-heuristic manner, where randomly-chosen features are perturbed accordingly. Our black-box query attack strategy is inspired by Rosenberg et al.'s [182]. However, our threat model considers *less information* about the target model under the black-box condition. We assume no access to prediction confidence scores or usage of sliding windows, so we use a modified version of their decision-based attack without these assumptions. Furthermore, if allowed by the dataset (see Section 4.4.6 later), our black-box attack strategy makes use of both feature addition *and* feature removal to increase the possible avenues for evasion. In our black-box query attack, a malware sample X is perturbed using features from a set of benign donor samples ($\mathcal{B}$) to generate an adversarial example X' using a transplantation-based approach. However, this is performed in a non-heuristic manner because of this attacker's weaker capabilities. This means that the feature to perturb during each iteration of the attack is chosen *randomly*, which can be added to (0 to 1) or removed from (1 to 0) the feature vector as permitted by the dataset. Feature removal can be performed when features absent in a benign sample (but perhaps functionally insignificant in X) are removed to cross the decision boundary. The attack process takes constraints regarding functionality preservation into consideration by only making perturbations that are allowed (see Section 4.4.6 later). The transplantation of the features continues until O is evaded, n_{max} is reached, or the possible features of the benign sample are exhausted (in which case another benign sample may be used).

4.3.2.2 Gray-box Query Attack

The gray-box attacker has access to the training data and knowledge of the statistical representation of the features across the dataset. Therefore, features can be added to a malware sample in a heuristically-driven manner using their frequency in benign samples (rather than randomly, as in the black-box scenario). In Figure 4.2, we present an overview of our gray-box query attack strategy.

In our strategy, features that are more frequent in benign samples are added to the malware sample X before others in order to promote traversal of the decision boundary as soon as possible. We show later that this significantly increases attack success and reduces queries to the oracle. Hence, as per Figure 4.2, from the data available to the attacker, the features from benign samples are sorted by their frequency across all benign samples into a vector $\vec{s}$ as a preliminary step to the attack — $\vec{s}$ can be reused whenever the attack is conducted. Then, given the ordered vector of features $\vec{s}$ (which contains features ordered by their frequency in benign samples), we can generate an adversarial example X' . That is, using a malware sample X , the next feature from $\vec{s}$ that preserves the original functionality of X is added to generate X' . Recall from before that only certain perturbations for each target platform will preserve the functionality of the malware sample (platform-specific details of valid perturbations in Section 4.4.6). As before, perturbations are validated for functionality preservation before being tested on O . The transplantation of features continues until the generated adversarial example X' evades O , n_{max} is reached, or the possible features are exhausted.

Preliminary step



The vector $\vec{s}$ can be reused whenever the attack is conducted.

Attack step

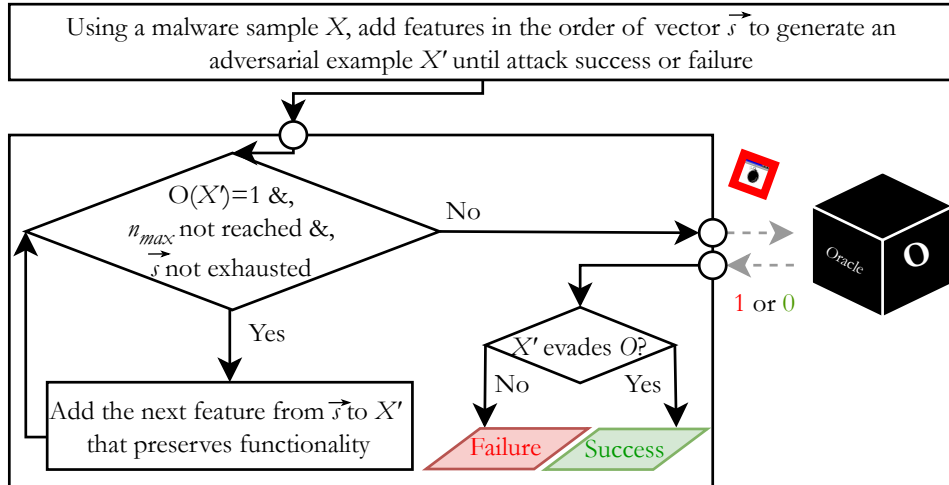


Figure 4.2: Overview of our gray-box query attack strategy.

We next provide details about the experimental setup we use for our evaluation of various moving target defenses against adversarial ML. This is followed by the study of various MTDs and other defenses under different attacker scenarios.

4.4 Experimental Setup

4.4.1 Datasets

As discussed in the previous chapters, sampling from the true distribution is challenging in many security applications and sometimes impossible. There is a lack of datasets that are up-to-date and publicly-available, which is a well-known problem in this domain [19, 20]. To partially mitigate this, in this chapter, we use three datasets that cover different platforms and collection methods that have been used previously (e.g., [223, 90, 66, 117, 168, 109, 23, 166]). Similar to the previous chapter, we use DREBIN for Android malware [18] and SLEIPNIR for Windows malware [8]. We additionally include AndroZoo for Android malware [11] as another dataset in our experiments. This enables us to demonstrate the consistency of our results and conclusions across the datasets, which transcend the characteristics and nuances of the datasets.

To recap, DREBIN has 123,453 benign samples and 5,560 malware samples, based on extracted static features. Hence, we use 5,560 samples from each class (benign and malware). SLEIPNIR consists of 19,696 benign samples and 34,994 malware samples. The features of this dataset are derived from Windows API calls in PE files parsed by the LIEF library [212] into a vector representation. SLEIPNIR is used to represent Windows out of simplicity with a convenient binary feature-space, enabling a clearer comparison between the Android and Windows datasets as our work is in the feature-space. To keep the dataset balanced, we use 19,696 samples from each class. Meanwhile, AndroZoo [11] is a large-scale dataset with timestamped Android apps from between January 2017 and December 2018. This dataset offers apps from different stores and markets, with VirusTotal summary reports for each. Similar to prior work [168, 166, 140], we consider an app malicious if it has 4 or more VirusTotal detections, and benign if it has 0 VirusTotal detections (with apps that have 1-3 detections discarded). Thus, the final dataset contains $\approx 150\text{K}$ recent applications, with 135,859 benign apps and 15,778 malicious apps. We use 15,778 samples from each class (benign and malware). As in recent publications [69, 90], and for completeness, we use a large number of features for each dataset, i.e., 58,975 for DREBIN, 22,761 for SLEIPNIR, and 10,000 for AndroZoo. Note, however, that we provide an experiment in Appendix C.1 with far fewer features for some datasets, suggesting that the number of features does not play a role in the results.

The datasets are split randomly for each class according to the Pareto principle, with an 80:20 ratio for training and test sets. Subsequently, using the same ratio, this training set is further partitioned into the final training and validation sets for constructing the models and defenses we evaluate. Effectively, this produces the 64:16:20 split that has been commonly used before (e.g., [120, 230]). The validation set is used to tune the models during development. The remaining test set is used in the attacks and is further split into training, validation, and the final test set. The training set here is used in the black-box transferability attack and corresponds to B_{train} , which is used to obtain the input-output relations of O for developing Δ . For

DREBIN, $|B_{train}| = 1423$. As SLEIPNIR and AndroZoo are larger, we perform additional splits to reduce $|B_{train}|$ to 1513 and 1494 respectively. Meanwhile, the malware samples in the final test set are the input samples for transferability and query attacks. For DREBIN, SLEIPNIR, and AndroZoo, there are 229, 230 and 234 such samples respectively.

Following the prudent practices detailed in Section 2.1.5, we consider the established guidelines for conducting malware experiments. For example, as the models in our evaluation decide whether an input sample is benign or malicious, retaining benign samples in the datasets is necessary. This also means that we do not need to strictly balance datasets over malware families. Instead, we balance datasets across the positive and negative classes and randomly select unique samples from each class to appear in the training and test sets (without any chance of repetition) [169, 8, 181].

4.4.2 Moving Target Defenses

In our work, we evaluate four MTDs that are configured to provide maximal robustness using the parameters suggested in their original papers. As we show in the next subsection, these MTDs offer sound performance in non-adversarial conditions for malware detection, so they are adequate for use in this domain. That is, these MTDs are domain-agnostic. We present the configuration for each defense in Table 4.1.

MTD	Parameters & Configuration
DeepMTD [200]	$w = 0.3, n = 20, T = 0.6$
Morphence [13]	$n = 4, p = 3, Q_{max} = 1000$
MTDeep [191]	5 DNNs as constituent models. Assumed $\alpha = 1$
StratDef	Variety-GT using same models as previous chapter. Assumed strong attacker and $\alpha = 1$

Table 4.1: Configuration of MTDs.

We next describe each MTD evaluated in this chapter at a high level. For a detailed literary comparison and review of the MTDs, we refer the reader to Section 2.4.1.

DeepMTD is a hybrid MTD [200]. n student models are generated by perturbing the weights of a base DNN, with w controlling the amount of perturbations. To make a prediction, if more than $T \times 100\%$ of the outputs from the student models are the same, the input sample is considered non-adversarial (and the majority class is the final prediction). Otherwise, it is classified as adversarial. To keep the system “moving”, new student models are generated when the system is idle.

Morphence is a hybrid MTD [13]. It generates n student models from a base model by shuffling weights randomly. Then, p student models are adversarially-trained (where $p \leq n$). For a user’s query, the prediction of the most confident student model is returned. Morphence uses a query budget (Q_{max}); student models are regenerated if the number of queries exceeds Q_{max} .

MTDeep [191] is a dynamic MTD. This defense models the interactions between attackers and defenders as a Bayesian Stackelberg game to produce a strategy vector to choose models at prediction-time based on attack

intensity. The resulting strategy can be pure (where only a single model is chosen) or mixed (where one of several models can be chosen).

StratDef is an MTD that we proposed in Chapter 3. It is a dynamic MTD that provides a framework to systematically construct, select, and strategize a set of models. StratDef gives particular consideration to the heterogeneity of its constituent models to minimize transferability, and it uses different optimizers to choose the best strategy for selecting a model at prediction-time.

4.4.3 Performance of MTDs in Non-Adversarial Conditions

Some MTDs in this chapter have not been applied to the malware detection domain before and instead have been first applied in the image recognition domain. However, similar to other defenses such as adversarial training (which was first applied to images and then to malware detection [90]), MTDs can operate effectively across a range of domains. As we show in Figure 4.3, the MTDs that we choose to evaluate offer sound predictions (90+% accuracy) on test sets in the malware detection domain, thereby showcasing their ability to work well in this domain too. DeepMTD seems to offer moderate performance considering AUC, but we still choose to include it in the evaluation just in case this is a result of the mechanism used against adversarial ML attacks, so we can compare it with other MTDs.

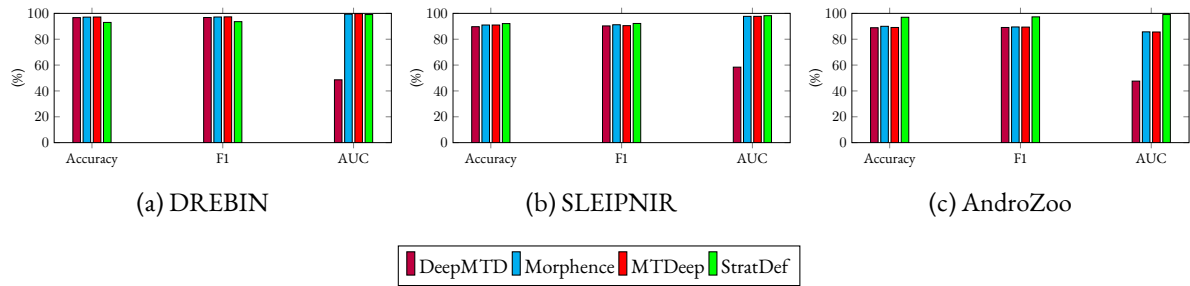


Figure 4.3: Malware detection performance of evaluated MTDs.

4.4.4 Other Evaluated Defenses

As MTDs are a type of *ensemble* defense — that utilize several models — we compare their performance with other types of defenses that use ensembles (in some manner) as baselines. These defenses have been featured in the previous chapter and have been reviewed in detail in Sections 2.3.1 and 2.3.2. Table 4.2 next provides the configuration for each defense followed by a description of each defense.

Voting is a static ensemble defense that has been previously applied to malware detection [194, 232]. As discussed in Section 2.3.2, multiple models are involved in making the prediction, which is determined by *majority* or *veto* voting. In majority voting, each constituent model makes a prediction, and the class with the most predictions is returned. In veto voting, if any constituent model returns a “malware” prediction, the entire system returns this prediction.

Defense	Parameters & Configuration
Majority voting	Using same models as StratDef-Variety-GT
Veto voting	Using same models as StratDef-Variety-GT
NN-AT	4 fully-connected layers (128 (Relu), 64 (Relu), 32 (Relu), 2 (Softmax)). Adversarially-trained with up 25% size of training data

Table 4.2: Configuration of other evaluated defenses that MTDs are compared with in our evaluation.

Ensemble adversarial training is a defense that trains a single model by using adversarial examples for other models [214]. The adversarially-trained model is then used to produce predictions. We use an adversarially-trained neural network (NN-AT) that is constructed using the training data described previously. Then, it is boosted by training with a quantity of adversarial examples that is 25% of the size of the training data, which is a sizeable amount to promote high adversarial robustness. Using a set of vanilla models (see Table 4.3), we generate adversarial examples by applying a range of attacks (listed in Section 4.4.6) and conducting the functionality preservation procedure (as described later). A proportion of these adversarial examples is then used for adversarially-training the neural network model (NN-AT) whose architecture is the same as the NN model in Table 4.3.

4.4.5 Substitute Models for Transferability Attack

For our transferability attack strategy (see Section 4.3.1), we use an ensemble of diverse substitute models to maximize evasion. With minimal tuning of hyperparameters to demonstrate the simplicity of our attack strategy, we construct four substitute models: a decision tree (DT), a neural network (NN), a random forest (RF) and a support vector machine (SVM) — see Table 4.3 for architectures. We use the scikit-learn [165], Keras [59] and Tensorflow [2] libraries for training. We use $\Theta_t = 0.75$ to ensure that an adversarial example evades the majority of substitute models before testing it on the oracle. We also test other values in Sections 4.5 and 4.6.

Model	Parameters & Configuration
Decision Tree	max_depth=5, min_samples_leaf=1
Neural Network	3 fully-connected layers (100 (Relu), 50 (Relu), 2 (Softmax))
Random Forest	max_depth=100
Support Vector Machine	LinearSVC with probability enabled

Table 4.3: Architectures of vanilla and substitute models. These models are used as the substitute models in our transferability attack strategies. Additionally, we construct a set of vanilla models with these architectures that are used to generate a set of adversarial examples for Ensemble Adversarial Training.

4.4.6 Generating Adversarial Examples

For the *transferability attack* (Section 4.3.1), we generate adversarial examples for substitute models using a variety of attacks: the Basic Iterative Method [116], Decision Tree attack [89], Fast Gradient Sign Method [87],

Jacobian Saliency Map Approach [160] and SVM attack [89] (see Table 3.3 for an overview of the applicable model for each attack). As these attacks produce continuous feature vectors and do not consider functionality preservation, we must perform additional steps to cater to these constraints, similar to Section 3.4.3. That is, after applying these attacks, we round the values in the generated continuous feature vectors to produce discrete feature vectors, representing the presence or absence of a feature (e.g., usage of a particular library). For example, if an attack changes the value of a particular feature to < 0.5 , it is set to 0 in the feature vector; meanwhile, if the value is ≥ 0.5 , it is set to 1 in the feature vector. We then check for invalid perturbations to preserve functionality within the feature-space. Only after invalid perturbations are reverted does an adversarial example proceed further in the attack pipeline according to the attack strategy.

For the *query attack* (Section 4.3.2), we apply the attack strategies under the black-box and gray-box scenarios. In both scenarios, a malware sample is perturbed by transplanting features from benign samples [182, 229, 168]. For example, if a particular feature is enabled in benign samples (i.e., its value is 1 in the feature vectors), it is added to the malware sample (changed from 0 to 1 in the feature vector for the malware sample) in order to move closer to crossing the decision boundary. The difference between the black-box and gray-box attack strategies lies in the choice of which features to perturb first. The gray-box attacker perturbs features based on their frequency in benign samples using their knowledge of the dataset. Meanwhile, the black-box attacker chooses which features to perturb randomly, as in [182], as no further information is available.

In both transferability and query attacks, the *permitted (valid) perturbations* (either feature addition or removal) for each dataset are determined by consulting industry documentation, previous work [124, 8, 168, 117], and the feature representation for each dataset. DREBIN and AndroZoo allow for both feature addition and removal [124, 3] (see Appendix A.1 for a summary of the allowed perturbations). In contrast, due to the encapsulation by LIEF’s feature extraction process when developing the dataset, we can only perform feature addition on SLEIPNIR. This results in a more *constrained attack surface* for SLEIPNIR, as there is a greater restriction on the perturbations that can be applied, leading to, as we will see later on, less effective attacks. Regardless, the procedure we use here offers a lower bound of functionality preservation within the feature-space, similar to prior work [192, 90, 124]. Recall that while we remain in the feature-space, the perturbations we perform could be translated to the problem-space as well. For example, feature addition could be achieved by adding dead-code or using opaque predicates [146]. Feature removal — which is more complex but still achievable — could be performed by rewriting the dexcode, encrypting API calls and network addresses (e.g., removing the features but retaining functionality).

4.4.7 Evaluation Metrics

We use several metrics in this chapter, similar to previous work [157], which we defined earlier in Chapter 2 and are clarified again here for convenience. The *evasion rate* is defined as the number of adversarial examples that evade the oracle over the number of adversarial examples that evade the substitute models. Furthermore, as MTDs may employ different models at prediction-time, an adversarial example may not always evade the oracle. Therefore, we use the *repeat evasion rate (RER)* as an additional metric for the evaluation. Recall that this measures the number of times an adversarial example evades the oracle out of 100 attempts. We also use

standard ML metrics such as accuracy, F1, AUC, and false positive rate (FPR) to evaluate the models and defenses under different system conditions.

We next present our experimental evaluation, where we evaluate MTDs and other defenses under different attacker scenarios. We begin with black-box results, followed by gray-box results and other experiments.

4.5 Black-box Results

In this section, we present our experimental findings for the black-box threat model, where we only have access to the predicted outputs. We first evaluate MTDs against the transferability attack strategies, followed by our query attack strategy.

4.5.1 Transferability Attack

In this section, we evaluate the effectiveness of the MTDs and other baseline defenses against the black-box transferability attack. Firstly, we analyze the results of the attack strategies overall, followed by an examination of how different attack strategies perform.

4.5.1.1 Evasion Rate

In general, we observe that attacks against the majority of MTDs are effective regardless of the attack strategy used. Figure 4.4 shows that MTDs seem easily evaded, with the average evasion rate across all attack strategies sitting at 56% for DREBIN, 12.9% for SLEIPNIR, and 50.3% for AndroZoo. However, the best attack performance is achieved by our attack strategy (Diverse Ensemble with $\Theta_t = 0.75$), with peak evasion rates of 96.3% for DREBIN, 42.3% for SLEIPNIR, and 96.1% for AndroZoo. This demonstrates that the majority of MTDs possess insufficient movement mechanisms and do not adequately prevent attackers from developing sufficient representations of them. This is especially observable in less constrained environments, such as with DREBIN and AndroZoo. For these datasets, the attack surface is greater due to the ability to perform more perturbations (i.e., both feature addition and removal), which leads to greater evasion. Meanwhile, for SLEIPNIR, there are greater restrictions on the perturbations that can be applied by attacks. Hence, this represents a more constrained environment for the attacker, where MTDs have it easier to defend themselves. The overall lower average evasion rate implies that MTDs can work well in these conditions.

In particular, we find that Morphence and veto voting are evaded the least, with an average evasion rate across all strategies sitting at $\approx 1\%$ for Morphence and 0% for veto voting. However, although veto voting offers good protection, it exhibits a high FPR (see Section 4.7 later). Interestingly, StratDef seems to be the least evaded MTD across all the datasets — for AndroZoo, the evasion rate of the attack sits at below 1%. While transferability attacks seem to be less effective for SLEIPNIR, we show later that query attacks are more effective and manage to extensively evade all defenses even with this dataset.

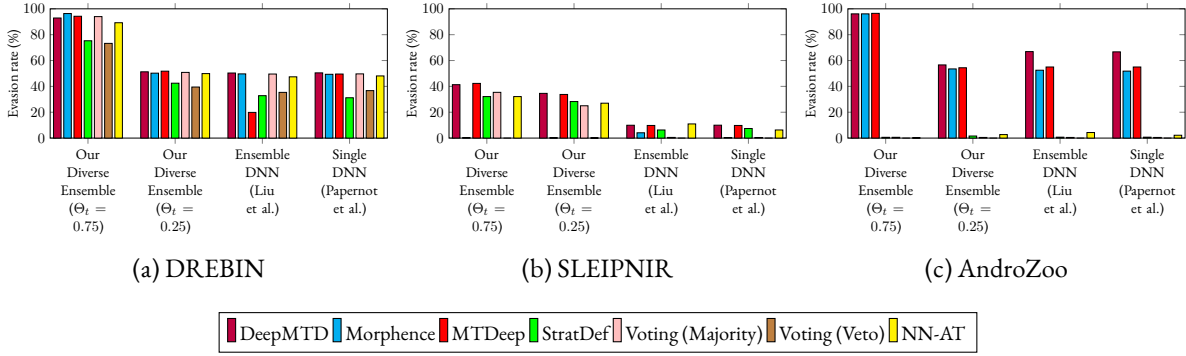


Figure 4.4: Evasion rate of black-box transferability attack strategies.

4.5.1.2 Comparing Attack Strategies

In terms of how the different attack strategies perform, Figure 4.4 shows that our attack strategy (Diverse Ensemble with $\Theta_t = 0.75$) achieves the highest evasion rate. For DREBIN and AndroZoo, the evasion rate sits at 65+% against most defenses (including single-model defenses), which is considerably greater than the $\approx 50\%$ evasion rate of the baseline attack strategies (Ensemble DNN and Single DNN). This is because, unlike other attack strategies, our attack strategy maximizes the transferability of adversarial examples locally across an ensemble of diverse substitute models before evaluating them on the oracle, thereby reducing attack failures and ensuring that adversarial examples are as transferable as possible, which is a strategy that is particularly suited to attacking MTDs. Even in a more constrained environment with SLEIPNIR, our attack strategies (using the Diverse Ensemble) surpass the Single DNN and Ensemble DNN attack strategies with a peak evasion rate of 40+% (versus $\approx 10\%$ for other strategies). As our attack strategy (Diverse Ensemble with $\Theta_t = 0.75$) offers the highest evasion rate across all datasets, we use this approach for the rest of the evaluation regarding black-box transferability attacks.

4.5.1.3 Varying Number of Input Samples, $|\Delta|$

We now vary the input samples to construct substitute models. This allows us to explore whether an effective transferability attack can be performed against MTDs with fewer samples to construct the substitute models. Recall that, previously, the maximum number of input samples was used to develop Δ (i.e., $|B_{train}|$). In this experiment, we therefore cap $|\Delta|$ and query each defense with different numbers of input samples from each class¹, up to the maximum available (1K+, i.e., $|B_{train}|$). The hypothesis is that as $|\Delta|$ increases, a better representation of the oracle (up to a point) should be produced, while fewer input samples should be less suspicious. We use our attack strategy (Diverse Ensemble with $\Theta_t = 0.75$) for this experiment.

Figure 4.5 shows that for DREBIN and AndroZoo, as $|\Delta|$ increases past two, the evasion rate increases and reaches up to 80+% after which it stabilizes for most MTDs. Generally, though, veto voting and StratDef are less evaded across both datasets. Fluctuations in the performance of some defenses are attributable to several

¹Therefore, Figure 4.5 starts with $|\Delta| = 2$ samples because a single sample from each class is used to build the training set for the substitute DNN.

reasons, such as model regeneration (Morphence) and dynamic model selection (StratDef). Moreover, Δ can become noisier due to inaccurate data, resulting in substitute models with a poorer approximation of the oracle’s behavior at some intervals, explaining the dips in evasion rates for some defenses (DeepMTD and veto voting). Interestingly, though, most defenses are no better than the single-model defense, NN-AT, in that the evasion rate of the attack remains similar for them.

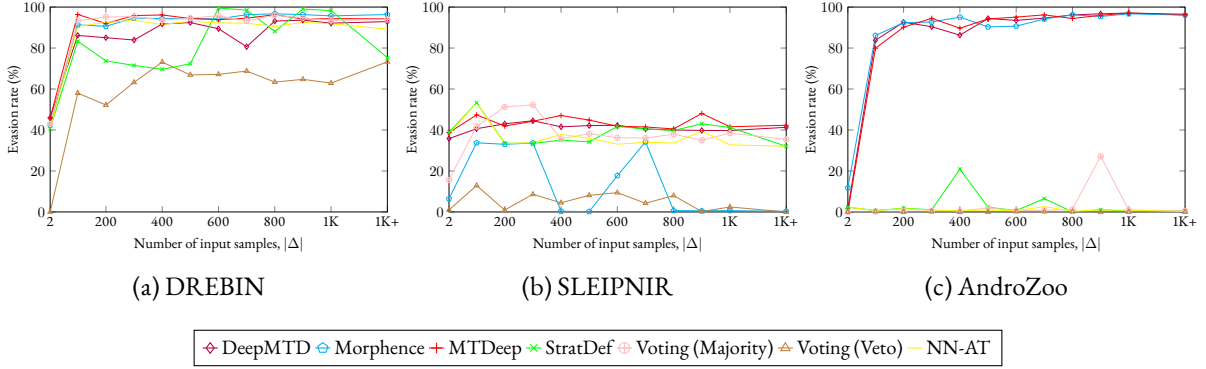


Figure 4.5: Evasion rate of black-box transferability attack vs. number of input samples (to produce the substitute model), $|\Delta|$.

Meanwhile, in the more constrained environment with SLEIPNIR, attack performance varies more, with the evasion rate peaking at 53%. However, there is significantly less evasion compared with DREBIN and AndroZoo due to a smaller attack surface being available to exploit. Morphence’s model regeneration procedure causes it to exhibit greater fluctuations in its performance under this more constrained dataset. Despite this, evasion is still achievable even if its query budget is exceeded. Meanwhile, for AndroZoo, we observe that nearly all MTDs are evaded in the same manner as for DREBIN. As $|\Delta|$ increases past 100, the evasion rate against these defenses sits at 80+%. Interestingly, StratDef and the non-MTDs (e.g., voting) continue to exhibit significant robustness, as in some cases, the attack fails to achieve any significant evasion.

4.5.1.4 Repeat Evasion Rate

Figure 4.6 shows the average repeat evasion rate (RER) versus the number of input samples of the attack from the previous section. Recall that this measures how many times (out of 100) an adversarial example evades the oracle, averaged across the entire attack. For StratDef, the average RER is 95% for DREBIN (minimum of 87%), 53% for SLEIPNIR (minimum of 44%), and 65.4% for AndroZoo (minimum of 0%). Meanwhile, for DeepMTD, Morphence, MTDeep, and voting, we achieve 100% RER, which means that adversarial examples have a higher chance of repeatedly evading the same oracle. This is either due to the predictive nature of each defense, the characteristics of their *movement* mechanisms, or the lack of diversity among their constituent models. As single-model defenses such as NN-AT are completely static, they experience 100% RER.

We next evaluate the effectiveness of the MTDs and other baseline defenses against the black-box query attack.

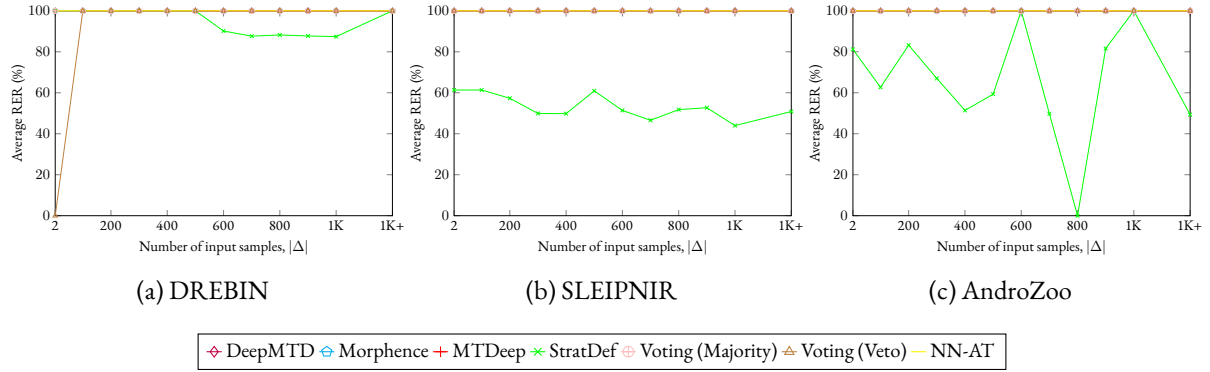


Figure 4.6: Average RER of black-box transferability attack vs. number of input samples. Lines appear merged as most defenses exhibit 100% RER.

4.5.2 Query Attack

Unlike the transferability attack, the black-box query attack does not use substitute models. Instead, a malware sample is perturbed using features from benign samples until it evades the oracle or the resources are exhausted (e.g., number of queries, available features to perturb). Figure 4.7 shows the results of the black-box query attack, clearly demonstrating that it outperforms the transferability attack, as models that were hardly evaded by the transferability attack (especially for SLEIPNIR and AndroZoo) are evaded by the query attack with a lower number of queries. We find that the average evasion rate of the query attack across all defenses sits at 72% for DREBIN, 69% for SLEIPNIR, and 90% for AndroZoo (with peak evasion rates of 99%, 91%, and 100% respectively).

Specifically, for DREBIN, veto voting and StratDef exhibit the greatest robustness, with the attack only achieving $\approx 20\%$ evasion rate. Meanwhile, for SLEIPNIR, although the environment is still more constrained in terms of the attack surface (i.e., fewer possible perturbations), veto voting and Morphence can be evaded still, unlike in the transferability attack. As before, the evasion rate is relatively lower for SLEIPNIR because there are fewer perturbations that can be applied to the malware samples, limiting the opportunities for evasion. However, as the perturbations used by the attack are tailored to this domain, we observe that the query attack yields better performance. This is especially evident in the results of the attack against StratDef for AndroZoo, where the black-box transferability attack yielded insignificant evasion. However, veto voting remains relatively less evaded across the datasets, likely because a single model can influence the prediction (even incorrectly) and the perturbations selected by the attack are insufficient to evade it. Though, as we show later, veto voting exhibits a higher FPR, which makes it less desirable as a defense (see Section 4.7 later). Meanwhile, defenses such as DeepMTD, MTDeep, and majority voting — as well as the single-model defense, NN-AT — offer minimal protection against the black-box query attack.

In addition, there is a strong correlation between n_{max} (the maximum number of queries allowed) and the evasion rate. We examine the number of queries in Figure 4.8, which shows the average number of queries required to evade each defense versus n_{max} . Most defenses can be evaded with less than 100 queries for DREBIN, 150 for SLEIPNIR, and 100 for AndroZoo (see Appendix C.2 for extended results). Typically, we

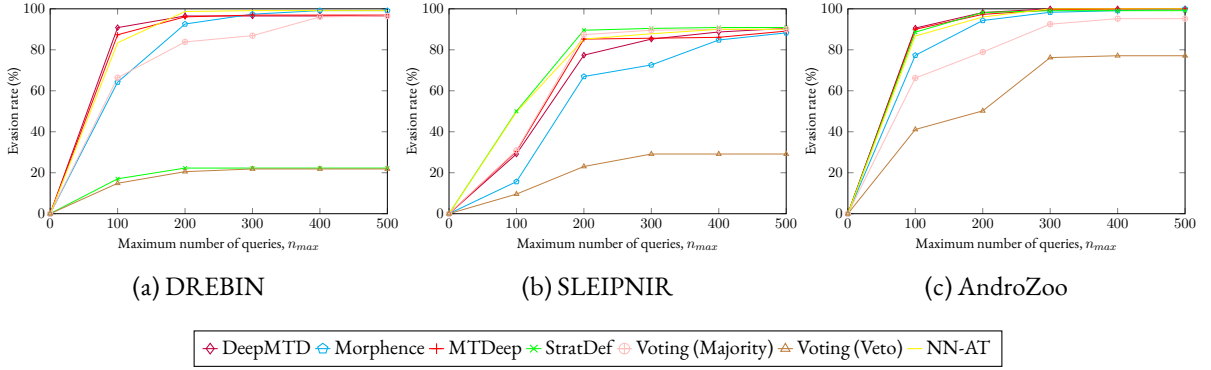


Figure 4.7: Evasion rate of black-box query attack strategy vs. maximum number of queries permitted.

find that attacks against static defenses (or those behaving statically) require fewer queries for evasion across all datasets. For other domains that use continuous features (such as image recognition), query attacks may require substantially more queries [50] to achieve attack success compared with domains that use discrete features. This is because perturbations in a discrete feature-space like ours (e.g., 0 to 1) have a greater effect on the final prediction, meaning that fewer of them may be required to achieve evasion compared with smaller perturbations made per query (e.g., $+0.01$) in a continuous feature-space.

Furthermore, experiments for the transferability attack showed that all defenses, besides StratDef, exhibit 100% average RER. For the query attack, the same is true for DREBIN — even for StratDef — where adversarial examples achieve 100% average RER against the defenses. For SLEIPNIR, the average RER for StratDef is 53.9% as the adversarial examples are not as effective as the transferability attack, while all other defenses exhibit 100% average RER. For AndroZoo, the average RER is 100% for all defenses for both attacks, except for StratDef, whose average RER sits at 53.5%.

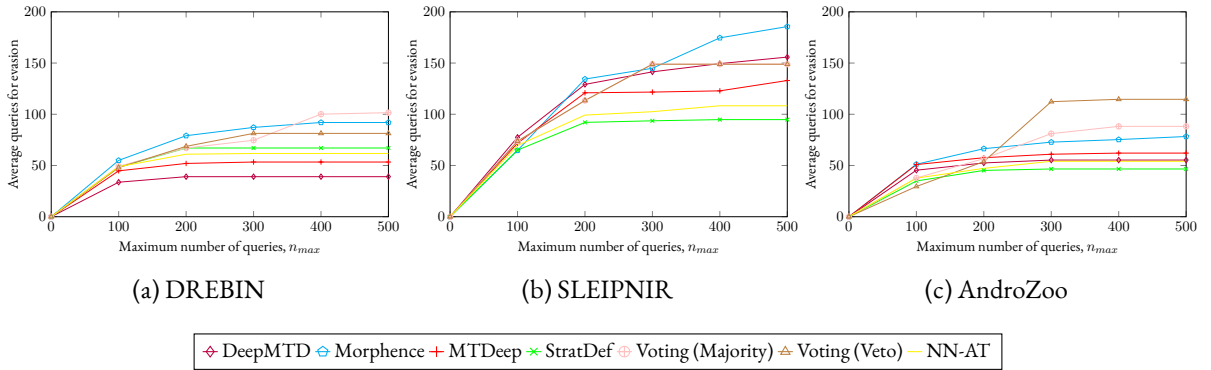


Figure 4.8: Average queries for evasion (black-box query attack) vs. maximum queries permitted.

Overall, in this section covering the black-box attacker scenario, we have demonstrated that transferability and query attack strategies can cripple most MTDs and baseline defenses, with our strategies performing most effectively. In the next section, we consider attacks under the gray-box threat model.

4.6 Gray-box Results

Under this attack scenario, we have access to the training data of defenses, which we use to develop substitute models for the transferability attack. Moreover, we have knowledge of the statistical representation of the features, which can be used to enhance the query attack. With greater information about the dataset, we also later evaluate each defense against Universal Adversarial Perturbations (UAPs) [143, 117].

4.6.1 Transferability Attack

We evaluate each defense against our attack strategy (Diverse Ensemble with $\Theta_t = 0.75$), having shown its superior performance. Our attack strategy, using the Diverse Ensemble but with $\Theta_t = 0.25$ is also included as an alternative, along with the Single DNN strategy [157] as a baseline for comparison. The Ensemble DNN is not included as it has been proven less effective than our attack strategy, and its performance is on-par with the Single DNN strategy. Recall that, under the gray-box scenario, the substitute models for each oracle (i.e., target model) are equivalent for each attack strategy as they are trained on the same training data as the defenses, rather than a synthetic dataset that is an estimation of O 's input-output relations as in the black-box version of the attack.

Figure 4.9 shows the evasion rate for the attack strategies under this threat model. In the less constrained environments with DREBIN and AndroZoo, our attack strategy can achieve a 55+% evasion rate across the MTDs for DREBIN and mostly 80+% for AndroZoo. Interestingly, when compared with the black-box transferability attack, the trends and the performance of each defense are similar. However, under a smaller attack surface and hence a more constrained environment with SLEIPNIR, there is greater difficulty in evading the MTDs in general. For the defenses that can be evaded, similar results are observable as in the black-box attack, with a 95+% evasion rate against the weaker defenses. In contrast, the vast majority of MTDs and defenses offer effective resilience against the attack, with defenses such as Morphence and veto voting remaining quite resilient to the adversarial examples. Furthermore, this attack is unable to evade StratDef, while the evasion rate for majority voting is non-zero, with less than 10 successful adversarial examples in total (which is why some results seem missing — see Appendix C.2 for full results).

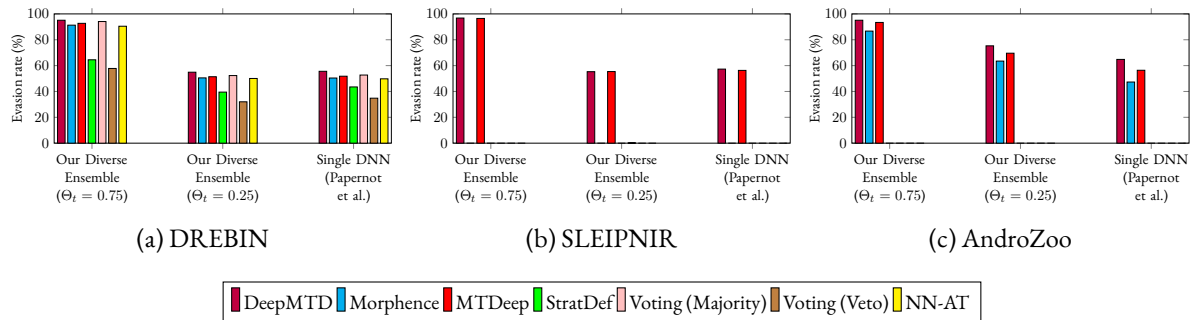


Figure 4.9: Evasion rate of gray-box transferability attack strategies.

In general, weaker defenses suffer more against the gray-box attack than the black-box attack, while for attacks against less static defenses, having access to the training data (as in the gray-box threat model) is not as beneficial for conducting attacks as it may seem. This phenomenon can be attributed to several factors. Firstly, substitute models for the black-box attack are more representative of each oracle, as they are based on direct queries to them when the synthetic dataset Δ is constructed. This suggests that, in these circumstances, the substitute models capture the oracle’s traits and behavior better, even with a smaller dataset. Consequently, adversarial examples for the black-box substitute models may transfer better to the oracle. Moreover, a larger training set for substitute models — as in the case of the gray-box transferability attack — may increase overfitting in the substitute models, resulting in inferior attack performance. For a set of experiments evaluating attack performance versus training set size, see Appendix C.3, where we discuss this in greater detail.

4.6.2 Query Attack

The gray-box query attack uses the frequency of features in benign samples to determine the order of transplanation. Compared with the black-box attacker, we hypothesize that this knowledge should reduce the number of queries needed to achieve evasion and also improve attack performance. Figure 4.10 shows the evasion rate versus n_{max} .

Similar to the black-box query attack, there is a strong correlation between n_{max} and the evasion rate, which reaches up to 99% for DREBIN, 94% for SLEIPNIR, and 100% for AndroZoo (with an average evasion rate of 84.6%, 91%, and 79.6% respectively, across all defenses and maximum query sizes). However, the average evasion rate of the gray-box query attack is also higher than the black-box query attack (by $\approx 20\%$) as well as the gray-box transferability attack, with nearly all defenses being evaded even in the more constrained environment of SLEIPNIR. As has been a common theme, defenses exhibiting mostly static behavior perform worse, with an evasion rate of 90+%. This shows that a heuristically-driven query attack method specifically designed for this domain can achieve better evasion of the MTDs, even in more constrained environments. Interestingly, veto voting still offers robustness for SLEIPNIR, although we achieve a higher evasion rate with this attack than previously seen.

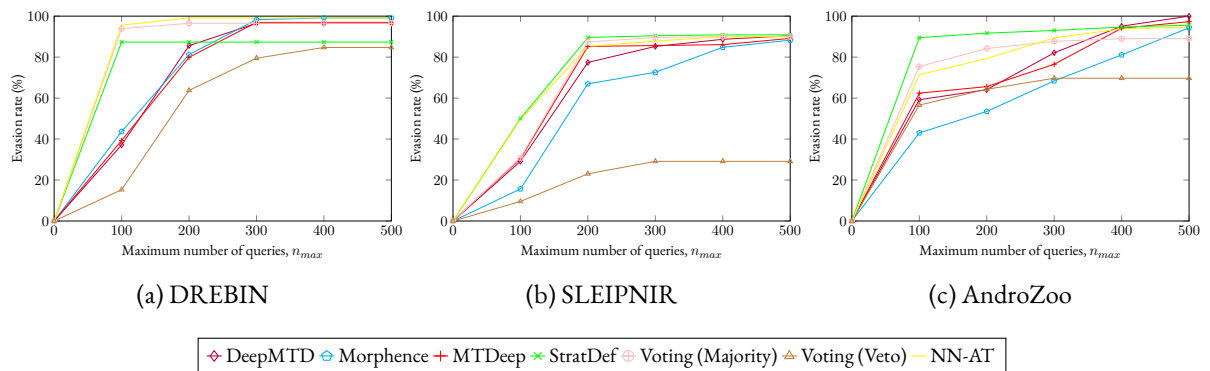


Figure 4.10: Evasion rate of gray-box query attack vs. maximum number of queries permitted.

The gray-box query attack also performs better than the black-box query attack with respect to the number of queries required to achieve this level of evasion. In Figure 4.11, we observe that fewer than 80 queries for DREBIN are enough to cripple most defenses. For SLEIPNIR, there is a further 44% decrease in the average number of queries compared with the black-box query attack. Meanwhile, with AndroZoo, there is also a reduction in the number of queries required to achieve evasion. Therefore, on the balance of numbers, the gray-box query attack outperforms the black-box attack considering the datasets and the number of queries permitted. Moreover, the average RER for the gray-box query attack for DREBIN is 100% for all defenses but is 0.2% higher than the black-box query attack on StratDef for SLEIPNIR. The average RER for AndroZoo similar to its black-box counterpart, with a 0.4% reduction for StratDef, but 100% for other defenses.

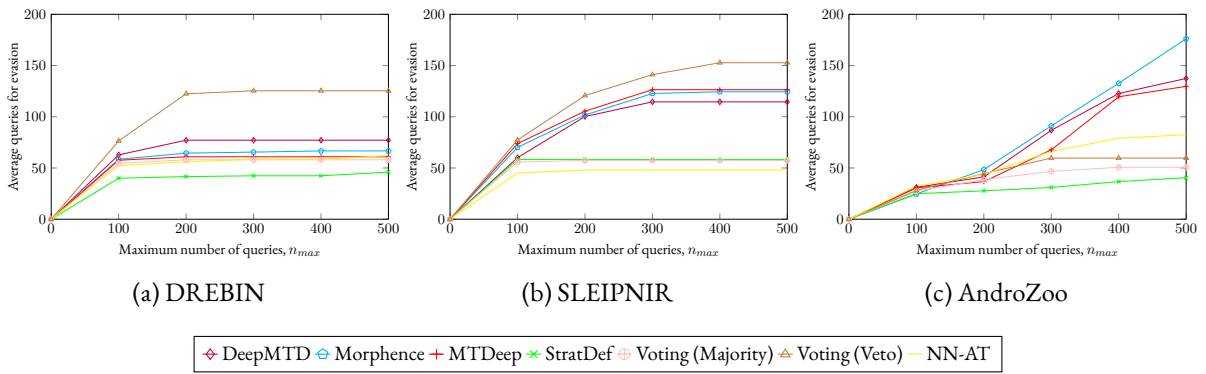


Figure 4.11: Average queries for evasion (gray-box query attack) vs. maximum queries permitted.

4.6.3 Universal Adversarial Perturbations

Recent research has identified universal adversarial perturbations (UAPs) as a cost-effective technique for generating adversarial examples [143, 117]. As discussed in Sections 2.2.5.2 and 3.5.4, with UAPs, a single set of perturbations can be applied to multiple malware samples to generate adversarial examples. In other words, sets of perturbations that are known to cause malware samples to cross the decision boundary are reused to generate adversarial examples. We evaluate each defense against adversarial examples generated with UAPs to compare with our attack strategies. The UAPs are derived from the dataset that the gray-box attacker has access to. Using the adversarial examples produced by the gray-box transferability attack, we examine if a set of perturbations has been reused exactly to produce adversarial examples from their original samples. In such cases, the adversarial examples are deemed to have been generated with UAPs.

Figure 4.12 shows the evasion rate of adversarial examples considered to be generated with UAPs. We observe that there is significantly less evasion compared with the transferability and query attacks, with an average evasion rate of 20% for DREBIN, 4.7% for SLEIPNIR, and 3.3% for AndroZoo (with peak evasion rates of 77%, 21%, and 14.1%, respectively). DeepMTD and MTDeep are evaded the most, especially for SLEIPNIR and AndroZoo. Nonetheless, similar to the gray-box transferability attack for these datasets, most defenses are not evaded by the adversarial examples generated through UAPs, with a significantly lower average evasion rate overall.

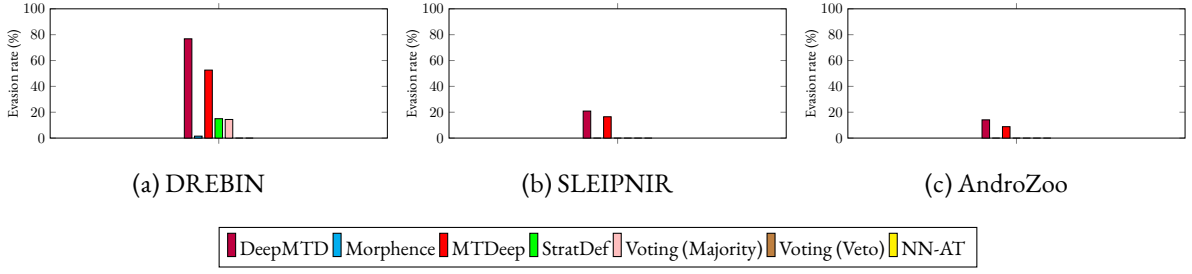


Figure 4.12: Evasion rate of attack (UAPs).

4.7 Beyond Adversarial Robustness

Some defenses consistently perform well against black-box and gray-box attacks. For example, defenses like Morphence, StratDef, and veto voting appear more resilient than others. Hence, it may seem appealing to deploy these defenses. However, metrics beyond adversarial robustness need to be considered, especially in the malware detection domain, where the false positive rate (FPR) must remain low [90, 202, 229, 125]. A high FPR means a less reliable and more frustrating service as analysts are flooded with false alarms and users find their benign queries misclassified as malware. We next evaluate the MTDs by simulating different system conditions with varying degrees of adversity to understand how each defense performs under these scenarios. In this evaluation, we consider additional metrics, such as accuracy, F1, AUC and FPR.

4.7.1 Procedure

In this experiment, we vary the adversarial intensity to ascertain the performance of the MTDs and other defenses in different system conditions. To do this, each defense is queried over 1000 times, with different proportions of adversarial examples represented by the attack intensity (q). As q increases, the system conditions become more hostile as the system faces more adversarial queries. For example, at $q = 0.5$, half of the queries to the oracle are adversarial, while the remaining proportion is an equal number of benign and non-adversarial malware input samples from the test set. The adversarial examples are those produced previously under the gray-box transferability attack to maintain uniformity, as the black-box attack produces different adversarial examples for each oracle. We evaluate each defense with $0.1 \leq q \leq 0.9$. We avoid the less likely cases of system conditions where $q = 0$ (where there are no adversarial queries) and $q = 1$ (where all queries are adversarial).

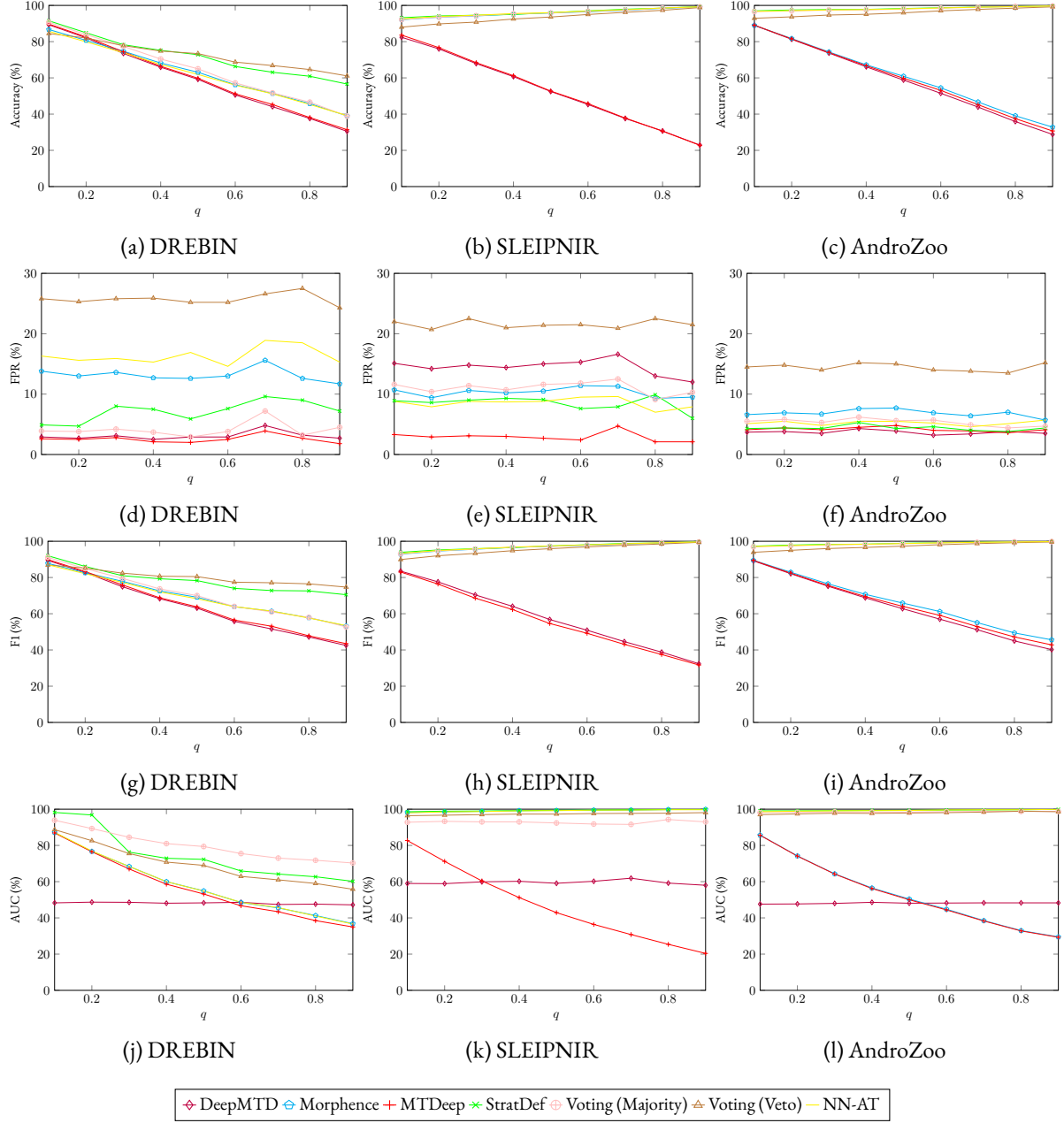
4.7.2 Results

Figure 4.13 shows the performance of each defense versus q following the evaluation procedure outlined before. We consider a range of evaluation metrics, such as accuracy, FPR, F1 and AUC. Across the Android and Windows datasets, both StratDef and veto voting appear to do well as the adversarial intensity increases as compared with other defenses. For example, at $q = 0.9$, the accuracy of StratDef is nearly 20% higher than the next-best MTD. Morphence also offers good accuracy for SLEIPNIR, but in the less constrained environments with DREBIN and AndroZoo, the results show that it can be evaded more in this attack setting. Meanwhile,

the accuracy of the other MTDs, DeepMTD and MTDeep, decreases significantly as q rises, indicating that they can be evaded more easily. Upon further analysis, their lower F1 and AUC support the conclusion that DeepMTD and MTDeep perform worse, especially at the higher adversarial intensities. For the Android datasets, Morphence also exhibits a lower ability to distinguish between classes as its AUC decreases more as q increases.

As we highlighted, the FPR is an especially important metric to consider in our domain. While veto voting and Morphence offer good accuracy, particularly for SLEIPNIR, the greater FPR associated with these defenses can also be seen across all values of q . Especially in the case of veto voting, its FPR is much higher than the other defenses across the datasets, making it likely that an individual constituent model of this ensemble defense makes an incorrect malware prediction for a benign input sample, leading to a false positive result. Morphence behaves similarly, as it uses the most confident model to derive the final prediction; hence, its higher FPR in some cases may also be attributable to this cause, as the most confident model may not make the correct prediction. If such defenses were deployed in a real-world setting, users would often receive incorrect predictions, which is rather unsuitable for the malware detection domain for the reasons previously enumerated. Conversely, the other MTDs tend to offer significantly lower FPR, especially MTDeep and StratDef. However, when all results are considered, it becomes clearer that MTDs like StratDef offer a more balanced performance for legitimate queries while also being more well-rounded against the adversarial threat.

We next explore whether MTDs can be fingerprinted and if reconnaissance activities to improve future attacks are possible. That is, if an attacker were to discover certain parameters of an MTD, they could enhance their attacks in the future and take any available information into consideration accordingly.

Figure 4.13: Accuracy, FPR, F1 & AUC of MTDs vs. q .

4.8 Fingerprinting & Reconnaissance

An enhanced understanding of defenses can be gained through fingerprinting and reconnaissance. Recall that reconnaissance can yield useful information for *future* attacks. If an attacker has a better understanding of how an MTD operates, they may be able to make their attacks in the future more successful. For example, discovering the query budget of a defense could allow an attacker to ensure that the target model behaves statically, making it less difficult to attack.

We next introduce two methods to demonstrate how this could be achieved, showing promising results and widening the scope for more elaborated attacks in future work.

4.8.1 Determining Predictive Nature of Defenses

As discussed in Section 2.3.2, an MTD is dynamic or hybrid in nature, while other defenses may be static. That is, an MTD may return different predictions for the same input sample, depending on the system conditions at prediction-time. On the contrary, defenses that exhibit static behavior (as well as hybrid defenses, to some degree), will return the same predicted outputs. Such defenses that remain static, fixed targets are ideal for an attacker, as they guarantee that an adversarial example can be trivially generated and reused, thereby ensuring its future success. Conversely, a defense behaving dynamically will exhibit less predictable behavior, leading to reduced repeat evasion (as seen before). Therefore, for attackers, understanding which kind of target model they are dealing with may allow them to tune their efforts accordingly.

Determining the predictive nature can be accomplished by querying the target model with different input samples repeatedly (n times). Subsequently, a consensus mechanism can determine whether predictions for the same input sample have changed across the queries. Fluctuations in the predictions are an indication of a dynamic defense, while static defenses will always produce the same prediction for a single input sample.

For instance, upon applying this technique with $n = 100$, it becomes clear that several defenses such as DeepMTD, MTDeep, and voting (majority and veto) produce predictions statically. This is understandable, as voting is static by nature. Meanwhile, the lack of variance in predictions for the MTDs indicates that DeepMTD may not be effectively regenerating student models, while MTDeep may be employing a pure strategy, which occurs when it computes the optimal strategy to be that only a single model from its ensemble serves the predictions. Perhaps, this is also why the evasion rate against these defenses has been found to be considerably higher than the other MTDs in our previous experiments. Meanwhile, Morphence also delivers predictions statically until its query budget is exceeded. However, as we show in the following section, if the attacker discovers that the oracle produces different predictions after n queries, they can adapt their attack to use fewer than n queries to ensure the oracle's static behavior. Conversely, StratDef exhibits dynamic behavior; for example, predictions for the same input samples vary by an average of 5.7% for DREBIN and 2.7% for SLEIPNIR on the test set described in Section 4.4.

4.8.2 Determining Movement of Hybrid Defenses

Hybrid defenses provide static predictions until a condition is met, at which point predictions for the same input sample may differ from before. For example, Morphence uses a query budget for this; once the budget is exhausted, a new set of constituent models is used, which could potentially differ in the predictions produced. However, the query budget may be discoverable using the method described previously to determine the oracle’s predictive nature. This is an example of stealing hyperparameters in ML [219].

Across the queries, the specific point where the oracle modifies its predictions can indicate whether a query budget exists and allow for an estimation of its value. For example, Morphence’s queuing system means that a longer waiting time implies greater system utilization. Therefore, when system utilization is low, we determine the predictive nature of the defense but with a single input sample and a much larger value of n (e.g., 10K). The intended result is that predictions vary across n to give an indication of the value of any query budget. If predictions do not change, it could be due to the input sample or because n is smaller than the query budget.

In an experiment to discover the query budget for Morphence (Q_{max}), Figure 4.14 shows that the prediction for the same input sample varies between 3K-5K and 7K-8K queries. This implies that the oracle (which is an example of a Morphence instance) changes after $\approx 1K$ queries (or $Q_{max} \leq 1K$), which can be confirmed by using another input sample. With this knowledge, an attacker could develop Δ for a black-box transferability attack without exceeding the query budget to ensure the static behavior of the oracle or perform a query attack with an n_{max} lower than the estimated Q_{max} . Recall that this is only an *evaluation instance* of Morphence. In practice, one may find that a different n is required to conduct this reconnaissance exercise.

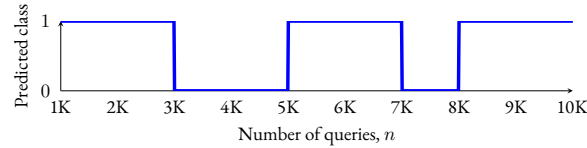


Figure 4.14: Prediction for the same input sample changes between 3K-5K and 7K-8K queries, implying that $Q_{max} \leq 1K$ in this evaluation instance.

4.9 Discussion & Recommendations

In this section, we discuss the overall findings from our evaluation. Based on these findings and our experimental results, we make key recommendations for developing effective MTDs against adversarial attacks in ML-based malware detection.

In some cases, the evaluated MTDs are no better than the other ensemble or single-model defenses. Among the evaluated MTDs, DeepMTD and MTDeep perform poorly across the board, regardless of the type of attack strategy. Our results show that both of these MTDs seem to exhibit a consistent lack of variance in predictions, leading to almost static behavior in practice. As already observed in previous work [171], DeepMTD’s mechanism for regenerating its student models only works when the defense is idle for some time. Regarding MTDeep, we observe that in this domain, its optimizer only generates pure strategies; that is, it only

uses a single model in practice to make the predictions. Therefore, these defenses are unable to take advantage of the main characteristic that an MTD should offer: dynamic behavior to increase the attacker’s uncertainty.

Meanwhile, Morphence and StratDef offer some degree of robustness in different attack scenarios. In particular, our results show that StratDef performs well for transferability attacks under both black-box and gray-box attack scenarios. Meanwhile, Morphence offers slightly better performance than other MTDs against query attacks, especially for lower values of n_{max} where the attacker faces more restrictions on the number of queries for their attack. The improved performance of these defenses can be attributed to more dynamic behavior (i.e., “how to move”). We observe in practice that both Morphence and StratDef vary their behavior dynamically and use higher model diversity (i.e., “what to move”) with diverse constituent models beyond DNNs (at least in the case of StratDef), which makes attacks against these MTDs less successful. Despite this, however, their performance is far from perfect, and in some threat models and attacks, particularly the gray-box query attack, both Morphence and StratDef can be well-evaded. Also, and particularly in the case of Morphence, we have shown that it can be fingerprinted, leaving key hyperparameters that govern its behavior exposed to attackers.

Yet, we believe that there is scope for MTDs to be a promising direction for defending against adversarial ML in the malware detection domain. If designed well, MTDs have the potential to introduce a layer of complexity and uncertainty for the attacker, helping to thwart attacks. Hence, based on our results, we next propose key recommendations for developing effective MTDs to motivate further work in this area.

Recommendation 1: Don’t use MTDs that behave statically

Static defenses have been found to be less effective in dealing with the adversarial ML threat when compared with more dynamic MTDs. Hence, we do not recommend using MTDs that behave statically. As we have shown, defenses with inadequate movement mechanisms are no better than single-model defenses such as adversarial training. In some cases, using a dynamic or at least hybrid MTD such as Morphence or StratDef seems to offer better performance across threat models and attack strategies in terms of evasion rates and other metrics. In addition, high repeat evasion rates can be prevented as adversarial examples work *inconsistently* against the oracle.

Recommendation 2: Increase the diversity of “what to move”

Using a dynamic or hybrid MTD may still be ineffective if its constituent models can be evaded through transferability. To minimize transferability between the constituent models, they need to be as diverse as possible. In fact, we have seen that MTDs that use different families (and not just DNNs) seem to perform better (e.g., StratDef). This means that other model families (e.g., random forests, support vector machines, etc.) should also be included in the constituent models beyond just DNNs. Hence, the diversity of “what to move” (e.g., constituent models) must be carefully considered and increased.

Recommendation 3: Be independent on “how to move”

The movement mechanism for MTDs should not be influenced solely by user behavior; movement should be independent. For example, some defenses (e.g., Morphence) regenerate models to “move” their configuration, which is impacted exclusively by user activity (e.g., number of user queries), rather than by an independent mechanism (e.g., time or randomness). A user-influenced mechanism makes the oracle an easier target as it can be manipulated by users. Therefore, MTDs must “move” independently and cycle through configurations to prevent attacks.

Recommendation 4: Consider leakage of hyperparameters

Hyperparameters are carefully designed and chosen for a specific context and may compromise a defense if leaked. For example, in Section 4.8, we demonstrate how a query budget can be discovered. This could make it easier for the attacker to consider the available query budget in a future attack, allowing them to take advantage of the defense before it changes its behavior. Therefore, it is critical that the design of an MTD accommodates the potential leakage of hyperparameters. The defender should operate under the assumption that hyperparameters may be leaked. For example, in the scenario described earlier, hyperparameter leakage could be mitigated by cycling through models at prediction-time (randomly or strategically) so that it cannot be reliably ascertained how often predictions are changing.

Recommendation 5: Greater system awareness is vital

While an MTD has several advantages, it cannot identify when an attack is taking place. For instance, this means that after several queries in query attacks, most MTDs can be evaded as single-model defenses (e.g., adversarially-trained models). Therefore, greater system awareness is vital, especially for protecting against query attacks given that particularly dynamic MTDs could vary their movement strategy at run-time if they were to detect an attack. Prior work (e.g., [52, 130]) has shown that stateful detection methods could help detect attacks against some DNNs. Therefore, such stateful detection methods as well as cyber-threat intelligence could be combined with an MTD to offer increased awareness and robustness. This would lead to an adaptive MTD system for detecting and reacting to potential *attacks in progress* (e.g., by changing strategies or deciding to make a different move). We explore stateful defenses extensively in the next chapter.

4.10 Conclusion

In this chapter, we studied the effectiveness of several moving target defenses against evasion attacks in ML-based malware detection for the first time. To conduct this evaluation, we used existing transferability and query attack strategies as well as novel strategies specifically tailored to MTDs that are designed to increase evasion. Under different scenarios, we demonstrated that MTDs are susceptible and vulnerable to evasion attacks, even with minimal queries to the target model. In some cases, the evaluated MTDs proved to be no better than the other ensemble or single-model defenses that were evaluated. Defenses such as Morphence exhibited a lower evasion rate for query attacks in some cases, while the evasion rate of the transferability attacks

against our defense, StratDef, was the lowest out of all MTDs. In fact, query attacks proved to be especially evasive. For example, our query attack strategies could achieve an evasion rate of over 90+% since the evaluated MTDs had no effective mechanisms for dealing with them.

Based on our findings, we provided key recommendations for developing effective MTDs in future work. If designed well, MTDs have the potential to introduce a layer of complexity and uncertainty for the attacker to thwart evasion attacks. Our recommendations emphasized that MTDs that behave statically should be avoided, while MTDs that utilize diverse constituent models with effective movement should be favored. That is, an MTD should use constituent models from different families (beyond just DNNs, even just different architectures) to maximize diversity for limiting the effectiveness of transferability attacks. Besides, these constituent models need to be used dynamically and moved strategically to maximize uncertainty and complexity for the attacker.

In addition, we recommended that MTDs be coupled with greater system awareness to detect and react to *attacks in progress*. This would pave the way for a promising way forward in dealing with query attacks. This may be accomplished by developing a response that is both automated and stateful [52], and/or based on cyber-threat intelligence [197, 238]. In particular, stateful defenses have been heralded as a promising lead. Some prior work (e.g., [130, 52]) has shown that they can detect attacks and prevent the generation of adversarial examples by monitoring and analyzing the sequence of queries received by the system. However, as the majority of the proposed stateful defenses have only been applied to the image recognition domain, we hypothesize that their effectiveness will be limited in our domain against adversarial attacks, as discussed in Section 2.4.3. This is due to the inherent differences in the malware detection domain in terms of how adversarial examples are generated.

Hence, in the following chapter, we present the first stateful defense against query attacks in the malware detection domain. We showcase a novel method for preventing adversarial query attacks and demonstrate its superior detection capabilities over prior stateful defenses in the first evaluation of its kind.

CHAPTER 5

STATEFUL DEFENSE AGAINST ADVERSARIAL QUERY ATTACKS

In this chapter, we propose a novel stateful defense for adversarial query attacks against ML-based malware detection, called *MalProtect*. Motivated by the evasion of many *stateless* defenses by query attacks observed in the previous chapter, we explore whether our proposed stateful defense can detect and reduce the effectiveness of adversarial query attacks in this domain as well as how it compares with prior stateful defenses in the first evaluation of its kind. Prior work has proposed stateful defenses to detect adversarial attacks that use similarity or out-of-distribution detection methods that may be effective in other domains. However, in the malware detection domain, as the methods to generate adversarial examples are inherently different, we hypothesize that such detection methods are ineffective, notwithstanding their limitations discussed in Section 2.4.3. We therefore propose a novel stateful defense, the first of its kind, that takes into consideration the constraints and characteristics of our domain to reliably detect adversarial query attacks using several detection techniques. The work in this chapter relates to OBJ4 and OBJ5, where we aim to propose a stateful defense and evaluate it in the context of other stateful defenses that have been proposed for adversarial ML.

5.1 Introduction

One of our key recommendations from Chapter 4 is to couple MTDs (or any prediction model) with greater system awareness in order to enhance the detection of attacks. Related to this, recent work has stressed that systems must monitor queries to identify hazards such as adversarial attacks [99]. In fact, ML-based anomaly detection has been mentioned as essential to detect the misuse of ML-based systems [37]. Hence, to detect adversarial query attacks, prior work has proposed *stateful defenses* that monitor the sequence of queries received by the system. Stateful defenses analyze these queries by examining various factors (e.g., for image recognition, this could be the similarity of feature vectors representing images).

In prior work, several stateful defenses have been proposed (e.g., [52, 130, 234, 110, 155]), however, they have a number of key limitations and have not been tested in the ML-based malware detection domain, where we have seen that query attacks can cripple a range of prediction models. Principally, the ML-based malware detection domain is significantly different from other domains, with more constraints imposed on attackers regarding the discrete representation of feature vectors and the preservation of malicious functionality when generating adversarial examples. Therefore, attackers may generate adversarial examples using techniques that are different from those in other domains [182]. Consequently, stateful defenses that have been applied to

other domains may be ineffective when their detection methods are used in our domain [99]. Additionally, prior stateful defenses have their own limitations, which we enumerated in Section 2.4.3, such as detection mechanisms that can be evaded, heavy dependence on account-based defenses, and poor maintenance of the query history. Together, these factors necessitate a new stateful detection approach for this domain.

Therefore, in this chapter, we present *MalProtect*, which is a model-agnostic stateful defense against query attacks in the ML-based malware detection domain. We aim to address the limitations of prior work and propose a stateful defense that operates effectively at detecting attacks and preventing the generation of adversarial examples in this domain. Upon receiving a new query, MalProtect employs several threat indicators that conduct an analysis to predict if there is an attack in progress. Across Android and Windows, we show that MalProtect reduces the evasion rate of query attacks by 80+%. We compare this to the meager performance of prior stateful defenses across a range of attack scenarios in the first evaluation of its kind. Additionally, we show that MalProtect produces interpretable decisions at a high level so that analysts can reason as to why the system made a particular decision. We also find that, despite being designed to protect against query attacks, MalProtect affords a degree of protection against other types of attacks, such as transferability attacks. Furthermore, we demonstrate that an adaptive attacker — with complete knowledge of MalProtect — fails to achieve significant attack success.

We next discuss the threat model used in this chapter, followed by a detailed description of our stateful defense, MalProtect.

5.2 Threat Model

Similar to the previous chapters, attackers aim to evade a feature-based ML malware detection classifier so that their malicious queries are misclassified as benign.

5.2.1 Target Model

The target model is a remotely-hosted malware detection system that predicts whether an input sample belongs to the benign or malware class. We refer to the classification system as the oracle O . The attacker aims to evade the oracle, similar to the prior chapters. Though, importantly, O is comprised of two components in this chapter: the underlying classifier F (i.e., the prediction model) and the stateful defense that protects F by analyzing queries to detect attacks (e.g., MalProtect — see Section 5.3 later).

To train F , input samples are represented as binary feature vectors using the features that are provided by a dataset. Recall that with the features $1 \dots M$, a feature vector X can be constructed for each input sample such that $X \in \{0, 1\}^M$, similar to previous work [174, 169, 90]. Within X , the presence or absence of a feature i is represented by either 1 or 0. As before, the features we employ in our work include, among others, libraries, API calls, permissions, and network addresses; these are provided by the datasets we use (see Section 5.4 later). With their class labels, several feature vectors can be used to train the binary classification model that we refer to as the classifier F (i.e., the prediction model). Then, when a user makes a query to O , a prediction is made

and returned. For the predicted outputs, we use 1 for the malware class and 0 for the benign class. The oracle O returns completely scoreless feedback [102].

5.2.2 Attacker's Goal

The goal of the attacker is to generate an adversarial example X' from a malware sample X to evade the oracle O and obtain a benign prediction. Suppose $O: X \in \{0, 1\}^M$ and we have a function $checkFunctionality()$ to check the functionality of X . Similar to previous chapters, we can summarize this as:

$$checkFunctionality(X) = checkFunctionality(X'); O(X) = 1; O(X') = 0 \quad (5.1)$$

5.2.3 Attacker Capabilities & Knowledge

We model three types of attackers for this work, which we have discussed extensively in Section 2.2.1. We consider the *gray-box attacker*, who has limited knowledge, with access to the same training data as the classifier F and is aware of the feature representation and the statistical representation of the features across the dataset. However, the attacker is unaware of F 's parameters, configurations, and other pertinent information. This represents a scenario similar to when some model data has been leaked. To perform query attacks, this attacker applies perturbations to the feature vector through a software transplantation-based approach [182, 168], guided by their knowledge of the dataset.

In contrast, the zero-knowledge *black-box attacker* can only observe the predictions received for their queries and has no information about the target model. However, they do have some information pertaining to the kind of feature extraction performed (e.g., the static analysis that a malware detection classifier may consider). This attacker also uses a software transplantation-based approach, but with considerably less information. Neither attacker is aware that MalProtect is a part of O .

We also consider an *adaptive attacker* (i.e., a *white-box attacker*) who has complete knowledge of the target model and knows that MalProtect is in place. Therefore, this more capable attacker tries to evade all components of the oracle O with a specific, tailored attack designed to evade MalProtect and the underlying prediction model.

We assume that all attackers possess ample computing power and resources to submit thousands of queries. Furthermore, unlike previous work [110, 52], we assume that attackers may control multiple user accounts and IP addresses, as in a Sybil attack [71, 228].

5.3 MalProtect

In this section, we provide a detailed description of our stateful defense, MalProtect. MalProtect is designed to protect ML models against adversarial query attacks that take place in the malware detection domain's feature-space. Figure 5.1 shows that MalProtect forms a layer of protection that monitors queries before they reach the prediction model F , which is the ML-based malware detection classifier to be protected. This allows MalProtect to detect adversarial queries that may evade the classifier (i.e., prediction model). The prediction

model F can be a vanilla ML model or a defense (e.g., an adversarially-trained model, an MTD, etc.). By having a defense as the prediction model used with MalProtect, it can offer *defense in depth*, as multiple layers of security can contribute to the overall security of the system.

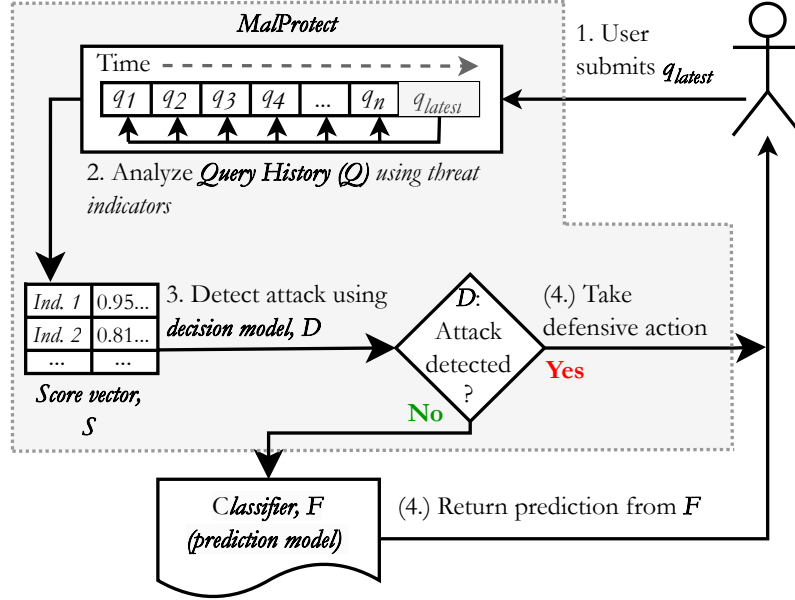


Figure 5.1: Overview of MalProtect.

When a user submits a query (Step 1), MalProtect employs several *threat indicators* to analyze the sequence of queries for attack detection (Step 2). This is similar to a system that makes decisions using data from multiple sensors [99]. The threat indicators use different criteria, principally driven by an anomaly detection-based approach. This includes examining aspects such as the autoencoder loss of queries, the distance between queries, the features shared across queries, and whether other characteristics of queries fit within different distributions. Based on the state of the query history, each indicator produces a score that reflects the likelihood of an attack in progress according to that indicator, with adversarial queries expected to cause higher scores. A *decision model* then takes the indicator scores as its input and predicts whether there is an attack in progress (Step 3). If an attack is detected, MalProtect takes some defensive action. If no attack is detected, the prediction model F returns a prediction for the user’s query (Step 4).

Not only do the scores produced by each indicator assist in interpreting how MalProtect arrives at a particular decision, but as we show later in Section 5.8, each threat indicator can be analyzed to determine its influence on the final decision. Next, we provide details about each of MalProtect’s core components.

5.3.1 Query History

MalProtect analyzes the query history to detect adversarial behavior. When a user requests a prediction for their query q_{latest} , it is appended to the query history Q . Queries can be represented more sparsely in the malware detection domain to reduce storage overhead, which has been cited as an issue in other domains [130].

The depth of analysis performed by MalProtect and the size of Q are controlled by the resources possessed by the defender when MalProtect is deployed, but, as we show later in Section 5.10, MalProtect needs very little spatial and computational resources to offer a very good performance. Unlike other stateful defenses that reset the query history once they reach capacity (e.g., [52]), MalProtect retains queries as a sliding window to maximize its detection capability. Finally, MalProtect does not distinguish between queries from different users, as an attacker can conduct a Sybil attack to evade account-based defenses [71, 228].

5.3.2 Indicators for Analyzing Query History

After adding q_{latest} to Q , MalProtect analyzes the query history using multiple *threat indicators* to detect attacks through an anomaly detection-based approach. In conventional anomaly detection, thresholds are employed to assess whether a sample meets the criteria for being abnormal. However, with MalProtect, we use a data-driven approach where the weights of each indicator are learned by a *decision model* to predict an attack. That is, instead of each indicator deciding if an attack is taking place, each indicator produces a score between 0 and 1 that reflects the likelihood of an attack according to its criteria. Scores are then aggregated and passed to a decision model that decides if an attack is in progress. This allows for a degree of interpretability too, as we can understand which indicators seem to play the biggest role in the decision model and which indicators produce the highest scores (giving clues about what is anomalous), as we show later. MalProtect is based on the principle that adversarial queries are expected to cause higher indicator scores as they are anomalous compared to legitimate queries and the training data distribution.

To conduct the analysis and produce scores, some indicators use standard anomaly detection techniques, such as the empirical rule [170], to assess whether a query fits the distribution of data. Meanwhile, other indicators use information about the training data to examine how much queries deviate from various measures. We next present a detailed description of each indicator:

Indicator 1: Distance between queries

This indicator assesses the similarity of q_{latest} to other queries in Q , which is the primary method adopted by some stateful defenses (e.g., [52, 130]). A smaller distance between queries is an indication of iterative perturbations during a query attack, as an attacker makes modifications to their queries to assess how the target model responds [52, 50].

This indicator therefore examines how anomalous the minimum distance between q_{latest} and other queries in Q is (represented by $\min Dist Q$). As the malware detection domain uses binary feature vectors (unlike other domains, e.g., image recognition), the L_0 distance is the most appropriate distance metric [45, 169], as it measures the number of features that are different between two feature vectors. Other norms such as L_∞ would always have a distance of 1 if at least one change was present between two feature vectors [169]. To assess the potential abnormality of $\min Dist Q$ without using a threshold and also to normalize it between 0 and 1, we put $\min Dist Q$ in the context of the average L_0 distance of all samples in the training data (represented by $avg Dist D$). If $\min Dist Q$ is significantly below this, it implies that q_{latest} is similar to another query in Q beyond the norm.

The score for this indicator is based on the percentage change between $minDistQ$ and $avgDistD$. A smaller percentage change means that there exists a query whose similarity to q_{latest} is significantly lower than the average of the training data ($avgDistD$). So that a higher threat is represented by a higher score value, we negate the result. Thus, the score for this indicator is calculated as follows:

$$S_1 = \frac{-1 \cdot (minDistQ - avgDistD)}{avgDistD} \quad (5.2)$$

Indicator 2: Enabled features shared across queries

This indicator assesses whether q_{latest} shares a significant number of enabled features with another query in Q beyond the norm. The rationale behind this is that an attacker could enable unused features in q_{latest} to intentionally increase the L_0 distance between queries in order to evade basic similarity detection (e.g., while trying to evade Indicator 1). However, some features must *always* remain enabled to preserve the original malicious functionality across a query attack. Thus, queries with the same enabled features would imply similarity specifically in this domain. Therefore, the highest number of enabled features shared between q_{latest} and another query in Q is calculated as $maxSharedQ$.

We then use the average number of enabled features shared between samples in the training data ($avgSharedD$) to assess to what extent $maxSharedQ$ may be abnormal. This is achieved by calculating the deviation of $maxSharedQ$ from this mean as a percentage change. A higher value indicates that q_{latest} shares an abnormally high number of enabled features with some other query in Q considering the training data's distribution. The score is calculated as follows:

$$S_2 = \frac{maxSharedQ - avgSharedD}{avgSharedD} \quad (5.3)$$

Indicators 3A & 3B: Number of enabled features

An attacker could rapidly traverse the decision boundary by enabling a substantial number of benign features in q_{latest} at once. An anomalous number of enabled features could indicate an attack attempt. Therefore, the Indicator 3A is the percentage change between the number of enabled features in q_{latest} ($|q_{latest}|$) and the average number of enabled features in training data samples ($avgFeaturesD$). A large percentage change implies that far more features are enabled in q_{latest} compared to $avgFeaturesD$. The score is calculated as follows:

$$S_{3A} = \frac{|q_{latest}| - avgFeaturesD}{avgFeaturesD} \quad (5.4)$$

However, the Indicator 3A only assesses whether $|q_{latest}|$ is anomalous considering the training data and does not compare with other queries in Q . It is also useful to assess whether $|q_{latest}|$ is anomalous considering the distribution of Q regardless of the training data, similar to previous works (e.g., [112, 22, 110]), as most queries are expected to be legitimate.

Therefore, we also use the Indicator 3B, which assesses to what extent the number of enabled features in q_{latest} is anomalous considering the queries in Q . The Indicator 3B uses the empirical rule (i.e., $3\text{-}\sigma$ rule)

[170], which is a standard technique in anomaly detection. The score is calculated as the percentage change from 3 standard deviations of the mean using the following equation, where $\mu_{FeaturesQ}$ is the mean number of enabled features of queries in Q and $\sigma_{FeaturesQ}$ is the standard deviation:

$$C = \mu_{FeaturesQ} + (\sigma_{FeaturesQ} \times 3) \quad (5.5)$$

$$S_{3B} = \frac{|q_{latest}| - C}{C} \quad (5.6)$$

Indicators 4A & 4B: Distribution similarity via autoencoder reconstruction loss

An autoencoder is a neural network where the input and output are the same but the hidden layers have fewer neurons [186]. This limits the amount of information that can travel through the model, requiring it to learn a compressed version of the input. Interestingly, input samples that belong to the distribution of the autoencoder's training data will produce a smaller distance between the input and output representations (known as the *reconstruction loss*) [156]. Thus, we hypothesize that an autoencoder trained on legitimate queries will produce a significantly higher reconstruction loss for an adversarial query. Other novelty detection techniques, such as one-class SVMs, have been found to be outperformed by autoencoders in this task in other domains [85].

Hence, Indicator 4A's score is the percentage change between the reconstruction loss of q_{latest} ($RecLoss_{q_{latest}}$) and the maximum reconstruction loss observed in training data samples ($maxRecLossD$). A higher percentage change would suggest a significant increase from the maximum reconstruction loss observed for training data samples, and thus an indication that q_{latest} may be adversarial. The score is calculated as follows:

$$S_{4A} = \frac{RecLoss_{q_{latest}} - maxRecLossD}{maxRecLossD} \quad (5.7)$$

As with Indicator 3B, we also consider the case where q_{latest} is only compared to other queries in Q regardless of the training data distribution. In this way, the Indicator 4B uses the empirical rule to assess whether the reconstruction loss of q_{latest} is anomalous considering the queries in Q . The score is calculated as a percentage change from 3 standard deviations using the following equation, where $\mu_{RecLossQ}$ is the mean reconstruction loss of queries in Q and $\sigma_{RecLossQ}$ is the standard deviation:

$$C = \mu_{RecLossQ} + (\sigma_{RecLossQ} \times 3) \quad (5.8)$$

$$S_{4B} = \frac{RecLoss_{q_{latest}} - C}{C} \quad (5.9)$$

5.3.3 Attack Detection

Once the scores are produced by each indicator, they are aggregated into the score vector S . The next step is to make a final decision using all the scores. For this purpose, static techniques can be used to combine several individual scores into an overall score or decision, such as a weighted sum model or a sum of squares model [79]. However, a key drawback of these static methods is that the weights for each indicator (i.e., how much

each indicator contributes to the overall score) must be selected manually by the defender. Moreover, static aggregation models cannot detect any underlying trends or patterns in the data.

Therefore, MalProtect instead uses a *decision model* (e.g., a neural network) that makes a prediction on the score vector S . Each threat indicator is a feature of this model, with the predicted output from the decision model representing whether there is an attack in progress constituting its decision. Importantly, MalProtect also allows analysts to understand why it has made a particular decision with its scoring system. Analysts can *further* understand the influence of each threat indicator on the final prediction by examining the global feature importance of the decision model. Later, in Section 5.4, we present the decision models that we evaluated, and we then discuss how each threat indicator influences the predictions made by these models in Section 5.8.

5.3.4 Defensive Action & Prediction

If no attack is detected by the decision model, q_{latest} is passed on to the classifier F (also known as the *prediction model*). The classifier makes a prediction of whether q_{latest} is benign or malware, which is then returned to the user. That is, if the latest query received is not considered to be part of an attack by MalProtect, the query is then forwarded to the ML classifier trained to decide whether an input sample is benign or malware. Note that the classifier can itself be made more robust using techniques such as adversarial training (as shown later). MalProtect only acts as a filter, which only lets what it considers to be *legitimate* queries to be passed on to the ML model used for malware classification.

If MalProtect does detect an attack, a defensive action can be taken, such as returning a specific prediction, returning inconsistent labels, notifying system administrators, banning user accounts, or rejecting further user queries. In the configuration of MalProtect that we use in this chapter, if an attack is detected, we take the defensive action of returning a *malware* prediction. This is as if the query had actually been forwarded to the classifier and that had been its output. This means that an attacker would encounter failure throughout the course of the attack, believing that their query is being consistently classified as malware, without necessarily knowing about MalProtect's presence. However, an intriguing line of future work would be to study what defensive actions could be more or less effective depending on the setting.

Having described our stateful defense in detail, we next provide information about our experimental setup, followed by our evaluation.

5.4 Experimental Setup & Preliminaries

5.4.1 Datasets

To mitigate the well-known problem of the lack of publicly-available and up-to-date datasets [19, 20] (see Section 2.1.5), we use three datasets representative of different architectures as well as collection dates and methods that have been used in a variety of studies (e.g., [223, 90, 66, 168, 109, 23, 166]). Like the previous chapter, the datasets we use are AndroZoo for Android malware [11], SLEIPNIR for Windows malware [8], and DREBIN for Android malware [18]. The consistency of our results across the datasets later show that MalProtect's performance transcends the characteristics and nuances of the datasets.

To recap, AndroZoo [11] contains Android apps from 2017 to 2018, offering apps from different stores and markets with VirusTotal summary reports for each. Similar to prior work [168, 166, 140], we consider an app malicious if it has 4 or more VirusTotal detections, and benign if it has 0 VirusTotal detections (with apps that have 1-3 detections discarded). The dataset contains $\approx 150K$ recent applications, with 135,859 benign and 15,778 malicious samples. To balance the dataset, we use 15,778 samples from each class. Meanwhile, SLEIPNIR consists of 19,696 benign and 34,994 malware samples. This dataset is derived from Windows API calls in PE files parsed by LIEF. As our work is in the feature-space, we use SLEIPNIR as a representation of Windows out of simplicity due to its binary feature-space. Recall that this permits a more precise comparison between the Android and Windows datasets. We use 19,696 samples from each class for this dataset. Meanwhile, DREBIN is based on extracted static features from Android APK files. The dataset contains 123,453 benign and 5,560 malware samples, from which we use 5,560 samples from each class. As in prior work [69, 90], and for completeness, we use a large number of features for each dataset, i.e., 10,000 for AndroZoo, 22,761 for SLEIPNIR, and 58,975 for DREBIN.

Similar to the prior chapters, the datasets are initially partitioned with an 80:20 ratio for training and test data according to the Pareto principle. This training data is partitioned again into training and validation data using the same ratio, producing a 64:16:20 split that has been extensively used in prior work (e.g., [133, 120]). We use the training data to construct the prediction models used in our evaluation. Meanwhile, we use a selection of randomly-chosen malware samples from a subset of the test data as the input samples for the query attacks. In total, there are 234 samples for AndroZoo, 230 for SLEIPNIR, and 229 for DREBIN.

Moreover, like the previous chapters, we consider the well-established guidelines and practices for conducting malware experiments [183], which were discussed extensively in Section 2.1.5. As our prediction models decide whether an input sample is benign or malicious, it is necessary to retain benign samples in the datasets. Furthermore, we do not strictly balance datasets over malware families but instead over the positive and negative classes. We randomly sample unique samples from each class to appear in the training and test data without repetition [169, 8, 181].

5.4.2 MalProtect Configurations

We develop two MalProtect configurations to showcase our defense’s capabilities. Each configuration uses a different decision model to predict an attack based on the indicator scores. For both configurations, we cap the query history size at 10,000. Later in Section 5.10, we discuss the efficiency related to this.

For each configuration, we follow the steps described in Section 5.3 for each dataset. For some indicators, we derive certain values from the dataset that are needed to assess queries (e.g., *avgDistD*, which is the average L_0 distance of samples in the training data). Then, to train each decision model, a defender could use real-world attack and score data. However, in its absence, a synthetic dataset can be used. Therefore, we develop a synthetic dataset of 1000+ labelled samples. For this, we initialize the query history to simulate past user activity with a randomly-chosen set of input samples from the training data. We then supply a range of queries (both legitimate and adversarial, as part of attacks) from our training data to have each indicator produce scores. This produces the synthetic dataset, where each threat indicator is a feature and the class labels

represent the true condition of the system at the time (whether it was under attack (1) or not (0)). To predict if an attack is in progress when given a score vector, the decision models we train are a *logistic regression model* for one configuration (MalProtect-LR) and a *neural network* for another (MalProtect-NN) (see Table 5.1 for architectures). In theory, both models should offer accurate predictions [72].

Model	Parameters & Configuration
Logistic Regression (MalProtect-LR)	No configurable parameters
Neural Network (MalProtect-NN)	4 fully-connected layers (128 (Relu), 64 (Relu), 32 (Relu), 2 (Softmax))

Table 5.1: Decision models for MalProtect configurations.

5.4.3 Other Stateful Defenses

To provide context for the performance of MalProtect, we compare it with three stateful defenses that have been initially tested in other domains to detect query attacks: the L_0 defense based on similarity detection using L_p norms (e.g., [130]); PRADA [110]; and Stateful Detection (SD) [52]. For a detailed literary review of each stateful defense, we refer the reader to Section 2.4.3. Although we keep the implementations of these stateful defenses as close to the original as possible (e.g., procedures and parameters), domain-specific techniques (e.g., encoding queries to generate a feature vector) are not applicable to the malware detection domain. Recall that stateful defenses themselves do not provide predictions; they protect prediction models. We provide the technical details of each stateful defense next.

The L_0 defense measures the similarity between feature vectors using the L_0 distance. This is akin to similarity detection schemes that use L_p norms in other domains (e.g., [130]). Recall that, as binary feature vectors are used in the malware detection domain, L_0 is the most appropriate measure of distance between two such feature vectors, X and X' [45, 169]. It measures the number of instances such that $X_i \neq X'_i$; a lower number of such instances implies greater similarity between two queries. For the L_0 defense, an attack is detected if there are queries with an L_0 distance of less than 10, as lower thresholds may miss attacks [130, 110, 52].

We also compare MalProtect with **PRADA** [110]. For PRADA, we use $\delta = 0.9$ for the threshold, as per the original paper [110].

Furthermore, we compare MalProtect with **Stateful Detection (SD)** [52]. For SD, the threshold for detecting an attack is derived by calculating the k-neighbor distance for the 0.1 percentile of the training set. At prediction-time, if the mean distance of the k-nearest queries falls below the calculated threshold, an attack is detected. For this, we use $k = 50$ (as per the original paper). The calculated thresholds for detection are as follows:

AndroZoo	SLEIPNIR	DREBIN
94.64749019607844	44.350758426966294	34.37145577151924

Table 5.2: Thresholds for detection for SD as per calculations based on original paper.

Recall that stateful defenses themselves do not provide predictions; they protect prediction models. Hence, we next provide detail about the prediction models we consider in our evaluation.

5.4.4 Prediction Models

In our evaluation, we use six *non-stateful* defenses as the prediction models. Rather than using vanilla models, we use single-model defenses, ensemble defenses, and moving target defenses as the prediction models to demonstrate their vulnerability to attacks despite being defenses in their own right. These are defenses that we have reviewed in Section 2.3 and that have been used in prior chapters. As explained in Section 5.3, by using MalProtect in conjunction with a defense as a prediction model (rather than a vanilla model), it can introduce defense in depth, which is a frequently-adopted strategy in information security.

For our evaluation, each prediction model is constructed according to the procedures outlined in their original papers using the training data for each dataset (see Table 5.3 for information about architectures). For single-model defenses, we use a neural network with defensive distillation applied (NN-DD) [161]. Additionally, we use several white-box attacks to develop adversarial examples for a set of vanilla models (see Section 5.4.6 later), which are used to adversarially-train a neural network (NN-AT). Similar to the previous chapter, we adversarially-train with a quantity of adversarial examples that is 25% of the size of the training data [214]. For ensemble defenses, we test majority voting and veto voting [174, 55, 194, 232, 174]. For moving target defenses, we evaluate Morphence [13] and StratDef (which we proposed in Chapter 3). Each prediction model is evaluated against attacks as is (i.e., a non-stateful setup). Then, each prediction model is combined with each stateful defense, allowing us to evaluate different setups.

Defense	Parameters & Configuration
Defensive distillation [161] (NN-DD)	Neural Network (128 (Relu), 64 (Relu), 32 (Relu), 2 (Softmax)) with defensive distillation applied.
Ensemble adversarial training [214, 174] (NN-AT)	Neural Network (128 (Relu), 64 (Relu), 32 (Relu), 2 (Softmax)). Adversarially-trained up to 25% size of training data.
Morphence [13]	$n = 4, p = 3, Q_{max} = 1000$.
StratDef	Variety-GT using same models as previous chapters.
Voting (Majority & Veto)	Assumed strong attacker and $\alpha = 1$. Using same models as StratDef.

Table 5.3: Architectures of prediction models.

5.4.5 Performing Query Attacks

Throughout the previous chapters, we have reiterated that when generating adversarial examples in the malware detection domain, the original malicious functionality must remain intact, and the feature vectors must remain discrete (i.e., consist of 0s and 1s) as each value within represents the presence or absence of each feature. Query attacks designed for other domains (e.g., [50, 51]) do not consider these constraints. Hence, they are incapable of producing effective adversarial examples in our domain, as we show later in an experiment in Section 5.7.

Instead, for our domain, query attacks can use software transplantation-based techniques to perturb the features of a malware sample using features from benign “donor” samples. In our evaluation, we modify query attack strategies that have been proven successful in this domain from prior work [182] as well as Chapter 4 (see Appendix D.1 for full details on attack strategies). Prior to starting an attack, we initialize the query history to simulate past user activity (as explained before). Then, using malware samples from our test set, we apply each attack strategy accordingly under each threat model to perturb malware feature vectors. We use the parameter n_{max} to govern the maximum number of allowed queries permitted, where the transplantation of features continues until the target model is evaded, n_{max} is reached, or the donor features are exhausted.

We apply a similar procedure as before in order to preserve the functionality of the original malware sample in the feature-space. For this, the valid perturbations for each dataset are determined by consulting industry documentation and prior work [124, 3, 123, 8, 168, 174]. If a feature has been modified invalidly — that is, the functionality has not been preserved in the feature-space — it is restored to its original value. This is to offer a lower bound of functionality preservation within the feature-space, similar to prior work [192, 90, 124]. Regarding the datasets and the permitted perturbations, AndroZoo and DREBIN permit feature addition and removal (see Appendix A.1). Meanwhile, for SLEIPNIR, only feature addition is possible because of the processing performed by LIEF to extract features when originally developing the dataset. While we remain in the feature-space, as we explained in previous chapters, the perturbations could be translated to the problem-space using techniques from previous work (e.g., [168, 182]). For example, feature addition can be achieved by adding dead-code or using opaque predicates [146, 168]. Feature removal — which is more complex — can be performed by rewriting the dexcode, encrypting API calls and network addresses (removing features but retaining functionality).

5.4.6 Performing Other Attacks

Prior work [130, 52, 234] has used *query blinding attacks* in image recognition with domain-specific techniques (e.g., modifying contrast or brightness). Such techniques are not applicable to our domain. Instead, we later present an adaptive attack strategy in Section 5.8, where the attacker has full knowledge of how MalProtect operates and aims to evade both it and the underlying classifier.

We also use a transferability attack to generate adversarial examples. These are used to adversarially-train models as a defender; and to evaluate stateful defenses later under different system conditions. The procedure for generating these adversarial examples is similar to the transferability attack strategy under the gray-box threat model from Section 4.3.1. To recap, we construct four vanilla models using the training data (see Table 5.4 for architectures) and apply a range of white-box attacks (BIM [116], Decision Tree attack [89], FGSM [87, 70],

JSMA [160] and SVM attack [89]) against these models to generate adversarial examples, in anticipation that they transfer to the target models.

Importantly, however, these attacks perturb features without considering the constraints of this domain. Therefore, we preserve the functionality of the original malware sample in the feature-space and discretize the generated feature vectors in a similar manner to the previous chapters. That is, once an adversarial example is produced, each value in the feature vector is discretized (i.e., if the value in the feature vector is < 0.5 , it is set to 0, else it is set to 1). Then, if a feature has been modified invalidly according to functionality preservation constraints, it is restored to its original value (similar to the process for query attacks). We then ensure that the adversarial example can still achieve evasion; if not, it is discarded following the attack pipeline described in Section 2.2.3 (and specifically in Figure 2.3). This procedure is designed to offer a lower bound of functionality preservation within the feature-space.

Model	Parameters & Configuration
Decision Tree	max_depth=5, min_samples_leaf=1
Neural Network	4 fully-connected layers (128 (Relu), 64 (Relu), 32 (Relu), 2 (Softmax))
Random Forest	max_depth=100
Support Vector Machine	LinearSVC with probability enabled

Table 5.4: Architectures of vanilla models.

In the following sections, we provide an extensive experimental evaluation covering the various attacker scenarios, where we evaluate MalProtect and compare it with other stateful defenses.

5.5 Black-box Query Attack Results

In this scenario, the attacker can only access the predicted output of the oracle. We conduct the black-box query attack against each *non-stateful* defense as is, to establish the baseline performance of the prediction models. We then evaluate and compare each combination of stateful defense and prediction model, including both MalProtect configurations.

Figure 5.2 shows the evasion rate of the black-box query attack versus n_{max} . Against non-stateful defenses (that is, the prediction models as they are), the black-box query attack can achieve a 100% evasion rate in some cases, with the average evasion rate sitting at 70+% across the datasets. These results clearly demonstrate that the non-stateful defenses do not provide adequate protection in this attack scenario, despite being defenses in their own right. Out of the prediction models, only veto voting limits the effectiveness of the attack to some extent, with the attack achieving a maximum evasion rate of $\approx 50\%$ across the datasets. Despite this, around one in two to three queries can still achieve evasion against this model.

Meanwhile, our stateful defense, MalProtect, significantly decreases the attack success, with peak reductions in the evasion rates for AndroZoo, SLEIPNIR, and DREBIN of 84%, 96% and 98%, respectively. This highlights the benefits (and necessity) of a stateful defense for this domain. Comparing the two MalProtect configurations, MalProtect-LR and MalProtect-NN exhibit comparable performance, with both configurations able to detect attacks after 5, 7, and 2 queries for AndroZoo, SLEIPNIR, and DREBIN, respectively.

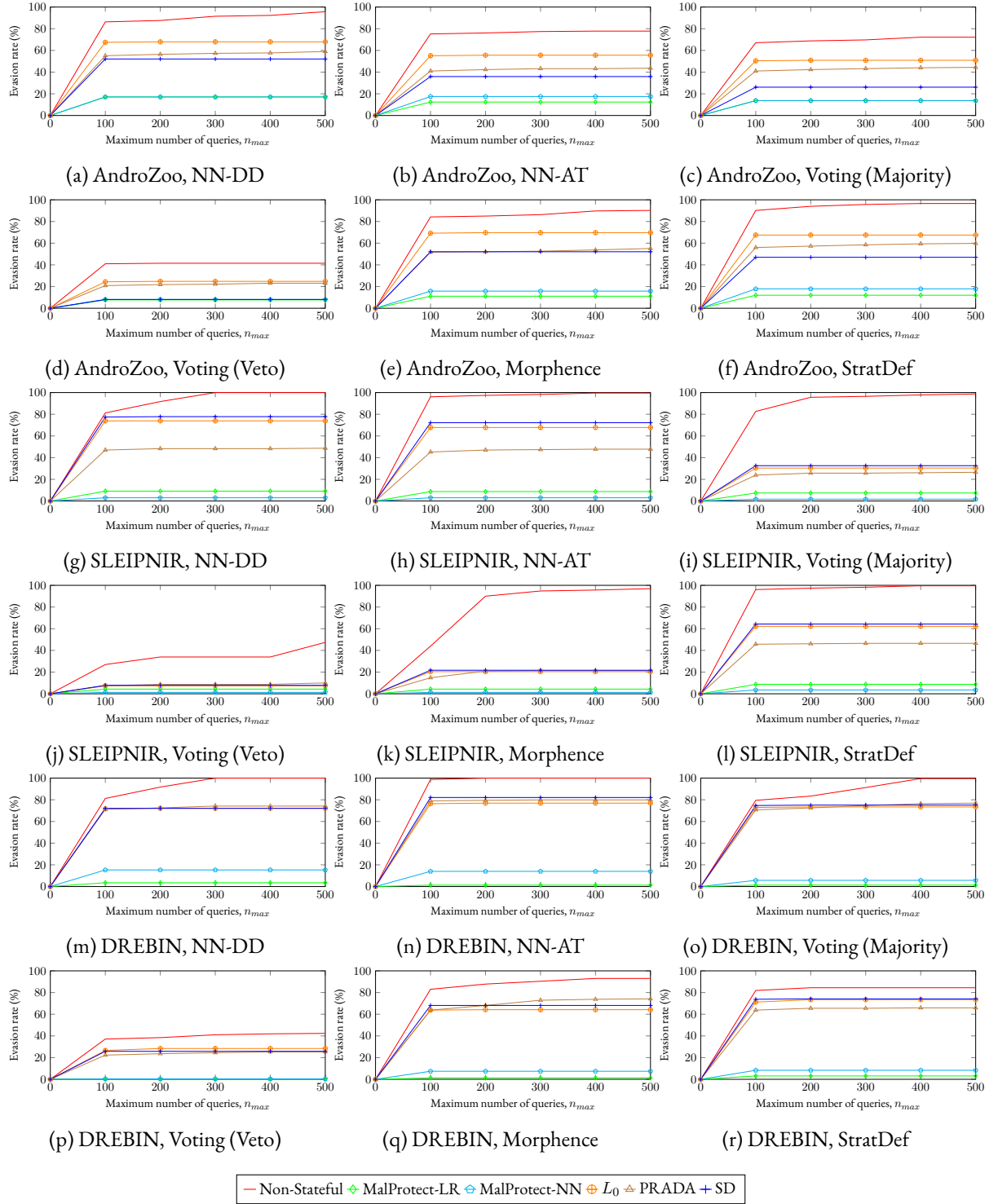


Figure 5.2: Evasion rate vs. n_{max} of black-box query attack against non-stateful defenses (prediction models) and each combination of prediction model and stateful defense.

This level of performance is achieved without greatly compromising other metrics, such as the accuracy or false positive rate on legitimate queries (as we show in Section 5.9 later).

Meanwhile, the other stateful defenses that we evaluate are significantly less effective against this attack. For AndroZoo, L_0 offers the mildest performance in terms of adversarial robustness, while SD offers slightly improved performance, but only marginally. In the attack’s weakest performance against veto voting, we find that SD performs similarly to MalProtect for AndroZoo and SLEIPNIR, but its performance is much inferior in other cases (with other prediction models) where MalProtect demonstrates robustness. For SLEIPNIR, PRADA offers better performance than the prior stateful defenses, though only marginally in most cases, while L_0 and SD seem similar in their defensive robustness. Meanwhile, for DREBIN, we observe that the three prior stateful defenses exhibit similar performance in all cases. That is, the black-box query attack can cripple most configurations involving prior stateful defenses, as they behave similarly to the non-stateful defenses. Overall, the black-box query attack can achieve a 60+% evasion rate in most cases against prior stateful defenses, compared with a peak evasion rate of $\approx 18\%$ for MalProtect. MalProtect provides demonstrably better protection against the black-box query attack.

5.6 Gray-box Query Attack Results

Prior work has stressed the importance of evaluating defenses against stronger adversaries [46]. Therefore, we next evaluate MalProtect’s performance against the gray-box query attack and compare it with other stateful defenses in that attack scenario. Under this scenario, the attacker has access to the training data of the prediction models and has more information about the features. The gray-box attack strategy therefore utilizes information about the frequency of features in benign samples to determine the order of transplantation, resulting in a more effective attack. As before, we conduct the attack against each non-stateful defense to establish a baseline performance, and then against each combination of stateful defense and prediction model.

Figure 5.3 shows the evasion rate of the gray-box query attack against all combinations of stateful defenses and prediction models, versus n_{max} . Against the non-stateful defenses, this stronger attack achieves significantly greater evasion across all the datasets, with 100% evasion rate in many cases. Perhaps the greatest difference from the black-box query attack can be observed in the results against veto voting, which was the least evaded non-stateful defense in Section 5.5. Whereas the black-box query attack peaked at $\approx 40\%$ evasion rate against this model (without any stateful defense), the gray-box query attack achieves $60 + \%$ evasion rate.

As far as protection by stateful defenses is concerned, MalProtect performs the best. In the best case for the configurations, it reduces the average evasion rate across all models from a peak of 84% to 13% for AndroZoo, 94% to 5% for SLEIPNIR, and 91% to 6% for DREBIN, while only taking $\approx 6, 6,$ and 2 queries to detect an attack for AndroZoo, SLEIPNIR, and DREBIN, respectively. Like the black-box query attack, MalProtect-LR and MalProtect-NN offer similar performance, with MalProtect-NN slightly outperforming MalProtect-LR in some cases and vice-versa. In fact, the gray-box query attack only achieves a maximum evasion rate of $\approx 20\%$ against MalProtect across the board.

Meanwhile, for the other stateful defenses, we observe trends similar to those seen in the results for the black-box query attack. For AndroZoo, SD outperforms L_0 and PRADA, with average evasion rates of 57.5%,

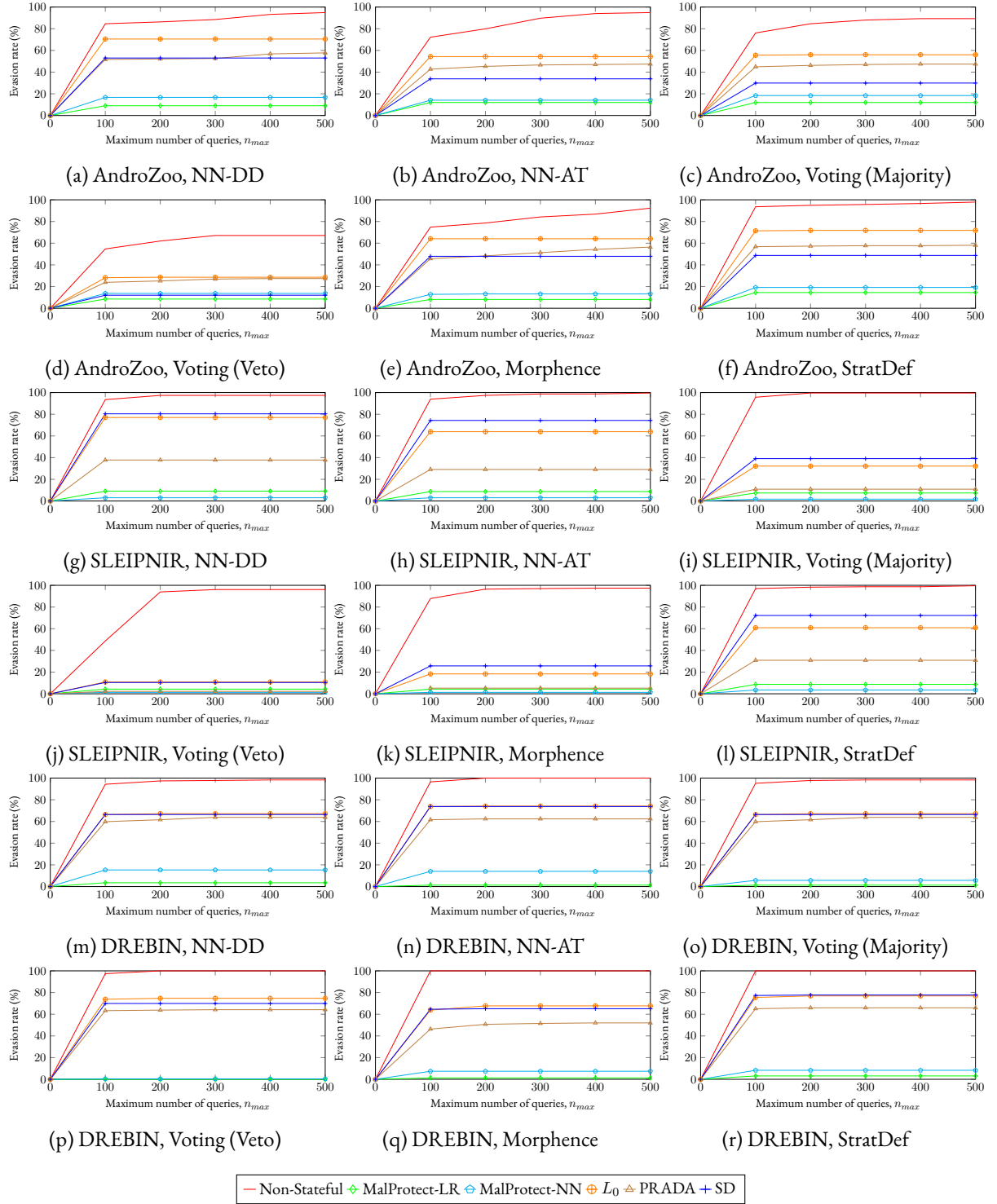


Figure 5.3: Evasion rate vs. n_{max} of gray-box query attack against non-stateful defenses (prediction models) and each combination of prediction model and stateful defense.

46.9%, and 37.5% against L_0 , PRADA, and SD, respectively. For SLEIPNIR, PRADA outperforms L_0 and SD for the majority of prediction models, a trend that was also observed in the black-box query attack results. Here, the average evasion rates of the attack sit at 43.8%, 19.3%, and 50.3% for L_0 , PRADA, and SD, respectively. Although PRADA appears to offer some robustness, it underperforms when other metrics are also considered (as we show later in Section 5.9). For DREBIN, the prior stateful defenses perform similarly with a more balanced performance. That is, they perform equally poorly, with the gray-box query attack able to achieve average evasion rates of 71.3%, 62%, and 69.9% for L_0 , PRADA, and SD, respectively, across all configurations. Interestingly, once again, PRADA (minimally) surpasses L_0 and SD in terms of robustness, though the poor general performance of these prior stateful defenses seen for both black-box and gray-box results reaffirms that relying on a single similarity or OOD detection mechanism is inadequate for this domain. In our domain, attackers use different techniques to generate adversarial examples that often include replacing features rather than making small detectable perturbations.

Further evaluating the evasion rate versus n_{max} , 100 or fewer queries are enough to achieve attack success in most cases. Query attacks in domains using continuous features (e.g., image recognition) may require substantially more queries [50] to achieve attack success compared with domains that use discrete features. This is because perturbations in a discrete feature-space (e.g., 0 to 1) have a greater effect on the final prediction, requiring fewer of them to accomplish evasion than perturbations made per query in a continuous feature-space (e.g., +0.01 to some feature). In fact, in the next section, we demonstrate the ineffectiveness of query attacks for other domains when applied to ML-based malware detection.

5.7 Evaluation of Additional Query Attacks

In this section, we also examine the Boundary [36], HopSkipJump [50], and ZOO [51] query attacks. We demonstrate their inability to generate adversarial examples in this domain, as these attacks are designed for other domains and do not consider the constraints of the malware detection domain (functionality preservation and discretization of features). We use each attack to try and generate adversarial examples against the NN-AT model. For this, once the feature vector of a malware sample has been perturbed, we discretize the feature vectors and evaluate whether the perturbations are permitted for each dataset. As usual, any invalid perturbations are reversed.

Figure 5.4 shows that these attacks are unable to achieve evasion *at all*. Evidently, the perturbations used to cross the decision boundary are reversed. That is, invalid perturbations are consistently made that must be discretized and reversed to ensure functionality preservation in the feature-space. Meanwhile, in Figure 5.4, we also show that our black-box query attack strategy achieves 80 + % evasion rate across the datasets.

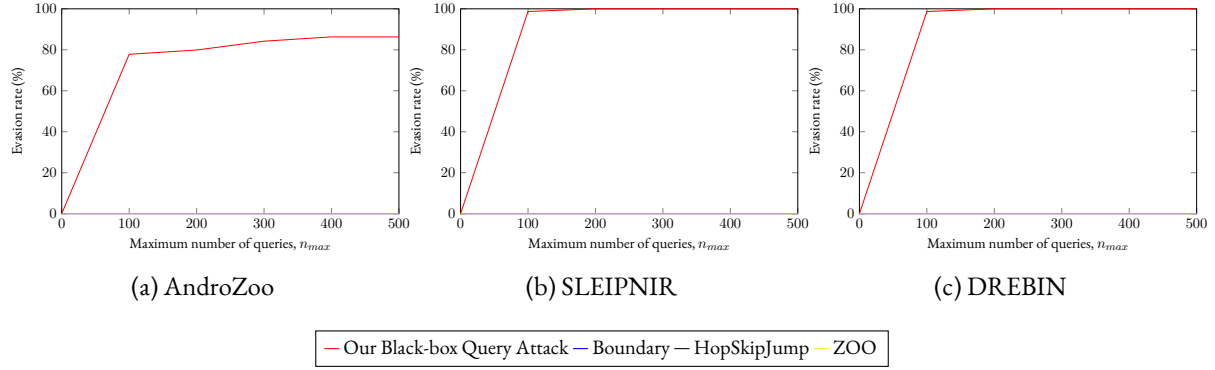


Figure 5.4: Evasion rate of additional query attacks against NN-AT model vs. n_{max} .

5.8 Interpretability & Adaptive Attack

5.8.1 Interpretability of MalProtect

As ML is being applied more to the cybersecurity domain, a key challenge is the interpretability of predictions made by models [147]. As an initial step towards addressing this in stateful defenses, MalProtect produces interpretable decisions. By observing the scores produced by each threat indicator, analysts can better understand how MalProtect made a particular decision, as higher scores for an indicator give clues about what is anomalous about queries.

In addition, the *influence* of each threat indicator on MalProtect’s decision can be examined by analyzing the *global feature importance* in MalProtect’s decision model. Recall that each threat indicator is a feature of the decision model. Therefore, we can assess the global feature importance using SHAP [132], which is a widely-used framework for interpreting ML predictions. Each feature of the decision model of the MalProtect configurations is given an importance value by SHAP at a global level, which produces the data observable in Figure 5.5.

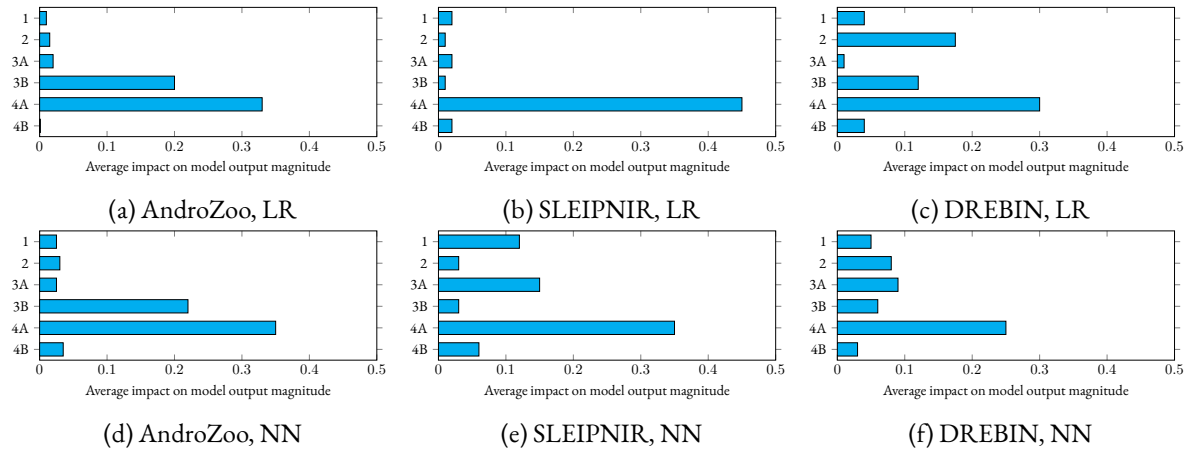


Figure 5.5: Influence of threat indicators in MalProtect (LR & NN) configurations.

This figure shows that the Indicator 4A has the greatest influence on MalProtect’s predicted output across all decision models and datasets, which indicates that a significantly high autoencoder loss is more likely to affect the predicted output. Meanwhile, the importance of the other indicators is more balanced. Intriguingly, the indicators based on similarity detection seem less influential, which is interesting because other stateful defenses rely solely on similarity detection to detect attacks. In malware detection, not only do we show that attackers can evade stateful defenses relying solely on such detection methods, but we further show that MalProtect rightly considers those indicators as less important.

Intriguingly, a capable attacker could leverage this information to evade MalProtect. In an adaptive attack (i.e., white-box attack) scenario [46, 215], the attacker has complete knowledge of MalProtect and, thus, knowledge of how much each indicator contributes to the final decision. With this information, an attacker could craft an adversarial example that evades both MalProtect and the underlying prediction model. We next examine how such an attack could be performed.

5.8.2 Attack Rationale

In the adaptive attacker scenario, the attacker knows precisely how MalProtect operates, its internal workings, and other pertinent information related to the threat indicators. At a high level, to evade MalProtect’s detection system, adversarial queries must appear as legitimate as possible and convince the indicators that the queries are distinct yet *normal*. Additionally, adversarial queries must remain within the distribution of legitimate queries. For example, we know that the Indicator 4A is the most influential in the final prediction, and therefore it would be essential to ensure that the autoencoder loss is not abnormally high. In essence, queries must remain far apart from each other while retaining their original malicious functionality and appearing legitimate. Importantly, of course, any generated adversarial example must also evade the underlying prediction model; otherwise, such an attack would be of no use. That is, the attacker must bypass all components of the target model to be successful [17].

5.8.3 Adaptive Attack Strategy

We modify the gray-box attack strategy to produce the adaptive attack strategy (see Appendix D.1). Firstly, we limit the number of features that can be perturbed in a single iteration of the attack so as not to exhaust possible perturbations early on. This frees up perturbations to be used in later iterations of the attack if earlier queries cannot achieve evasion. While this may increase the query distance and make queries appear less anomalous, other indicators (e.g., Indicator 4A) may still be able to trigger attack detection. To deal with this, the adaptive attack strategy also *removes* a proportion (p) of features at each iteration. This means that queries will be more distant, with fewer shared and enabled features, while conforming to the distribution of legitimate queries and the training data. For example, a *combination of features* that may cause an increase in the autoencoder loss (such that it appears anomalous) may be removed. Only the AndroZoo and DREBIN datasets support the removal of features while preserving functionality within the feature-space. Therefore, the adaptive attack strategy can only be applied to these datasets.

5.8.4 Results

Figure 5.6 shows the evasion rate of the adaptive attack against the stateful defenses (averaged across the maximum number of queries permitted, up to 500) versus the percentage of features removed (p). Successful adversarial examples evade each stateful defense as well as the underlying prediction model, which is NN-AT.

For both datasets, the adaptive attack fails to achieve significant increases in the evasion rate against either MalProtect configuration. In fact, in some cases, the average evasion rate decreases as p increases. This is likely because the (benign) features selected to cross the decision boundary are in fact removed. The adaptive attack causes other stateful defenses to exhibit similar performance to the gray-box query attack, with an average evasion rate of 72.9%, 61.9%, and 67.7% for L_0 , PRADA, and SD, respectively, with a consistent evasion rate across different values of p . This is expected, as the adaptive attack is specifically designed to target MalProtect and not necessarily the other stateful defenses (which are only included as a comparison here).

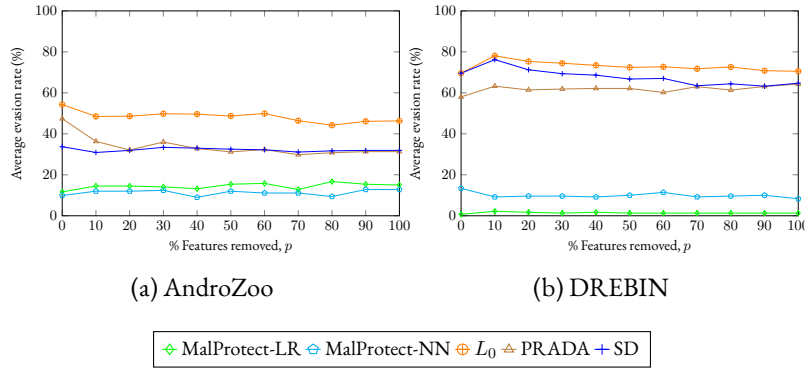


Figure 5.6: Average evasion rate vs. p for adaptive attack. The average is across the maximum number of queries permitted.

5.9 Beyond Adversarial Robustness

We next evaluate the stateful defenses under different system conditions using metrics beyond the evasion rate in a similar manner to Section 4.7. To do this, we query stateful defenses with benign and malicious queries, some of which are adversarial. This is imperative, as the original task of classifying queries as benign or malicious must also be done properly.

5.9.1 Procedure

We examine the performance of stateful defenses in different system conditions with different levels of adversity. For this, we generate adversarial examples using the transferability attack strategy described in Section 5.4.6. Each stateful defense (with NN-AT as the prediction model) is then queried 1000+ times with values of $0.1 \leq k \leq 0.9$, where k represents the adversarial intensity. For example, at $k = 0.1$ — which represents a less adversarial environment — 10% of all queries are adversarial examples, while the remaining 90% are equally split between benign and *non-adversarial* malware samples from the test set. As k increases, the system conditions

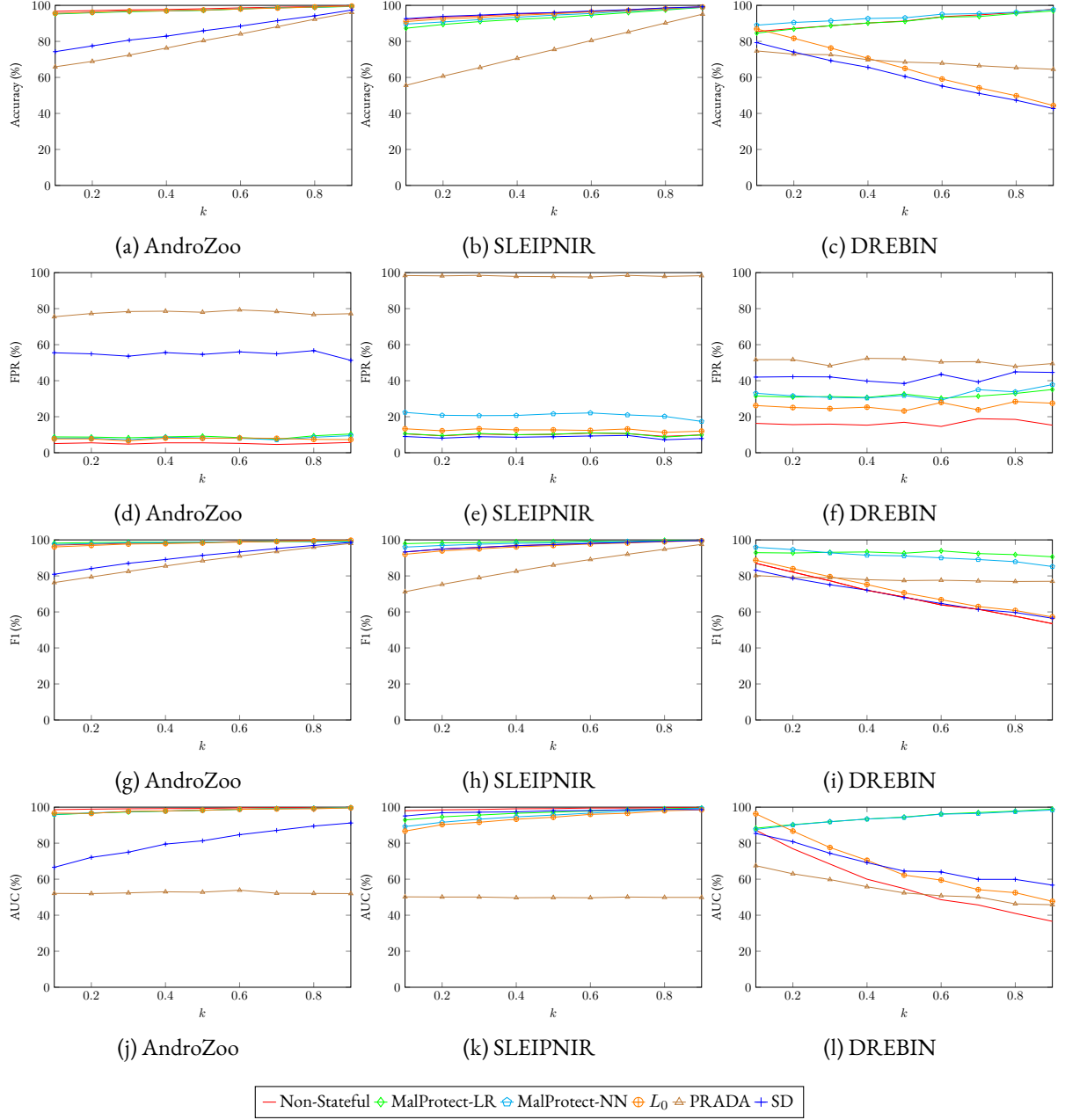
become more hostile as the system faces more adversarial queries. In this experiment, we do not consider $k = 0$ (no adversarial examples) or $k = 1$ (only adversarial examples), as such environments are less likely. As before, the query history is initialized as necessary with random samples from the training data.

5.9.2 Results

Figure 5.7 shows the performance of each stateful defense versus k . In particular, MalProtect exhibits reasonably stable performance across the datasets considering all metrics, with the accuracy and F1 remaining greater than 90%, demonstrating MalProtect’s ability to properly classify user queries. Intriguingly, other stateful defenses also appear to provide decent performance in terms of these metrics in some cases. For AndroZoo and SLEIPNIR, L_0 , PRADA, and SD all exhibit relatively stable accuracy and F1 as k increases. This is possibly because the stateful defenses have it easier to detect the (less evasive) adversarial examples, which constitute most of the queries, as they are generated by attacks not for this domain. Recall that the perturbations applied by the attacks to generate the adversarial examples may be reversed if features are modified such that functionality is affected; this would limit attack success. Regardless, these stateful defenses do not perform as well as MalProtect. For DREBIN, we observe the decline of the non-stateful NN-AT model and stateful defenses except MalProtect when these metrics are considered.

Importantly, though, additional metrics must be examined to obtain a comprehensive assessment of performance. Analysis of the AUC reveals MalProtect’s superiority in distinguishing the benign and malware classes across the datasets without disrupting the classification of legitimate queries. Meanwhile, PRADA and SD exhibit significantly lower AUC in some cases (especially as k increases), implying that their decisions are no better than flipping a coin, which is contradictory to the higher accuracy and F1 that they exhibit. This is possibly because as k increases, the number of malware queries also increases in the test set because of more adversarial queries. If PRADA, for example, decides that most queries are adversarial (and therefore malware), this would lead to high accuracy, even if it misclassifies benign queries. Therefore, PRADA seems less capable of separating between the classes in this domain. This is further evidenced through an examination of the false positive rate (FPR) next.

In the malware detection domain, as we have explained before, a low FPR is essential to ensure a reliable service for users [90, 202, 229, 125]. Although a stateful defense may improve protection against attacks, the FPR may also increase compared with the underlying prediction model. This can be attributed to the sensitivity of detection mechanisms; that is, benign queries may be more frequently misclassified as malicious. Importantly, though, MalProtect offers a low FPR that is as close as possible to the non-stateful prediction model for all datasets as it uses several indicators to predict an attack. Meanwhile, L_0 is the only other stateful defense that offers comparable performance to MalProtect in this regard. However, as we have seen, it does not work well at defending against query attacks in this domain. Conversely, PRADA exhibits a significantly high FPR for nearly all datasets, which supports the notion that it is merely classifying most queries as malware rather than actually distinguishing queries between the classes in this experiment. This is somewhat resembled by SD, with its high FPR for the Android datasets. Overall, across the metrics and datasets, we showcase MalProtect’s ability to perform well across a range of different queries.

Figure 5.7: Accuracy, FPR, F1 & AUC vs. k for each stateful defense under different adversarial intensities.

5.10 Overheads of MalProtect

We have shown MalProtect’s ability to defend prediction models against adversarial query attacks. However, assessing the storage and time costs of deploying MalProtect is also crucial. MalProtect’s detection capacity and performance are proportional to the defender’s available resources; with additional resources, MalProtect can store and analyze more queries in Q rapidly and precisely to detect attacks.

5.10.1 Storage

The storage cost scales linearly with the size of Q . In our experiments, the size of Q is capped at 10,000, which only consumes ≈ 0.08 MB storage for all the datasets. MalProtect’s storage costs are therefore negligible considering modern resources.

5.10.2 Prediction Time

There is latency associated with analyzing the query history and returning the prediction to the user. For MalProtect, the worst-case prediction time — which is when it must compare a query with all of Q — scales linearly with the size of the query history. In our experiments where $|Q|$ is capped at 10,000, the worst-case prediction time sits at ≈ 0.4 seconds for AndroZoo and SLEIPNIR, and ≈ 0.6 seconds for DREBIN. For MalProtect-LR and MalProtect-NN, the analysis stage is identical, while obtaining the prediction from the decision model is near-instant, which produces the same overall cost for both configurations. While other stateful defenses such as L_0 , PRADA, and SD sometimes have lower worst-case prediction times (≈ 0.03 - 0.6 seconds across the datasets), MalProtect takes more time to perform an in-depth analysis of the query history, which provides more protection against attacks. As we have shown, other stateful defenses are more susceptible to attacks when applied to this domain.

5.10.3 Optimization

MalProtect can be optimized with GPUs. We find that running MalProtect on an NVIDIA A100 GPU reduces the worst-case prediction time to ≈ 0.21 - 0.41 seconds across the datasets when $|Q|$ is capped at 10,000.

5.11 Conclusion

In this chapter, we presented MalProtect, which is the first stateful defense for adversarial query attacks in the ML-based malware detection domain. As we have shown, ML prediction models and defenses exhibit significant vulnerability to query attacks in this domain. Prior stateful defenses that have been applied to other domains provide meager protection when applied here as well, notwithstanding their other limitations. Meanwhile, our stateful defense, MalProtect, does not rely solely on a single form of similarity or out-of-distribution detection, as these prior defenses do. Instead, several threat indicators and a decision model are used to detect attacks more effectively. Our evaluation has shown that MalProtect performs well against attacks under various scenarios and also offers more reliable predictions for non-adversarial queries than prior stateful

defenses for Android and Windows datasets. Furthermore, MalProtect displays resilience even against adaptive attackers.

An interesting and open research direction is to examine whether cyber-threat intelligence [197, 238] can play a role in detecting attacks. A key characteristic of malicious and adversarial queries is that they may reuse features to evade a defense. Hence, a system could be trained to detect such instances. We expand on this in further detail in the next chapter, where we also discuss other intriguing research directions.

CHAPTER 6

CONCLUSIONS & FUTURE WORK

In this chapter, we provide concluding remarks for the work presented in the dissertation. This is followed by a discussion about interesting future research directions.

6.1 Conclusions

Machine learning has become a popular solution for a variety of problems. For malware detection, ML-based systems surpass rule-based and signature-based detection systems not only in terms of accuracy but also in their unmatched capacity to extract patterns and infer from vast amounts of data. However, the introduction of machine learning in this domain has not come without its challenges. From a security perspective, the attack surface has been widened, allowing attackers to evade classifiers through adversarial ML attacks. For example, in an evasion attack, a truly *malicious* software executable can be carefully constructed so that it is misclassified as *benign*. To remedy this issue, several defenses against adversarial ML have been proposed. However, as we have thoroughly discussed, these have a number of limitations, such as their ineffectiveness when applied to our domain, reduced performance on clean data, and the inability to detect attacks.

This dissertation has therefore proposed novel defensive methods against adversarial attacks in ML-based malware detection using moving target defenses and stateful defenses. We have demonstrated the ability to reduce the evasion rate of attacks under different threat models and attacker scenarios with our proposed defenses. In addition, we have developed a more thorough understanding of the various types of defenses against attacks that have been proposed in recent years. Our work has evaluated, characterized, and compared the performance of prior defenses with our own proposed defenses. When taken as a whole, this dissertation has contributed novel defensive approaches against adversarial attacks and provided a fresh perspective on the various methods for defending ML-based malware detection models.

In Chapter 2, we laid the groundwork for appreciating the work in the dissertation. We introduced the fundamental concepts and core ideas related to machine learning and its applications in cybersecurity. Specifically, we discussed the inner workings and anatomy of ML-based malware detection. Therein, we identified the challenges posed by adversarial machine learning, especially evasion attacks, as we enumerated a variety of threat models and attack scenarios. We also analyzed prior efforts that aimed to develop defenses for adversarial ML, further motivating the work in this dissertation. We provided a detailed literature review of the

most related work to ours, identifying the key limitations of prior work and promising research directions to further motivate our work.

In Chapter 3, we proposed a novel defense, StratDef, based on a strategic moving target defense approach. The work in this chapter was related to our research objectives OBJ1 and OBJ2. We investigated how a strategized moving target defense could provide better protection against adversarial attacks in the malware detection domain. We suggested methods to combat attacks strategically based on an MTD approach — rather than relying on a single model — by considering various factors that had not been explored in detail before, such as model heterogeneity, threat level, and information available about the attacker. Furthermore, we investigated the various dimensions of a strategic MTD, such as what, how, and when it should adapt itself given the current environment it is operating within. In a comprehensive evaluation, we showed StratDef’s ability to increase accuracy by 50+% and reduce the success of targeted attacks across Android and Windows malware. To provide context for StratDef’s performance, we conducted an evaluation of several other models and defenses that correctly considered the constraints and characteristics of this domain under different threat models and attacker scenarios. We demonstrated the fragility of the vast majority of these existing defenses, and StratDef’s ability to mostly outperform them, motivating the need for a strategic defensive approach. While this work demonstrated that an MTD approach could successfully operate in this domain, especially against transferability attacks, several research avenues presented themselves. First, an intriguing avenue was to evaluate and compare various moving target defenses for adversarial ML. Additionally, it remained to be seen whether stateful defenses had any scope to defend against attacks in this domain.

In Chapter 4, we next delved into the characterization and evaluation of several moving target defenses for adversarial ML. The work in this chapter was related to our research objective OBJ3. Prior to our work, the performance of MTDs against adversarial ML attacks under different threat models had not been evaluated, nor compared with each other, nor compared with other existing defenses in great detail. It was imperative to assess whether MTDs are a line worth investigating further to remedy adversarial ML attacks, and if so, what kinds of strategies and ways to design MTDs are more effective. Therefore, we conducted the first comprehensive study and evaluation of moving target defenses that were applied to the malware detection domain. To conduct this evaluation, we used attack strategies from prior work. Additionally, we proposed novel transferability and query attack strategies that increased the evasion rate by 50+%, especially against MTDs. Our evaluation showed that the evaluated MTDs were generally ineffective, as we could achieve significant evasion (90+% evasion rate in many cases) with minimal effort and resources, and even in complete black-box scenarios. Interestingly, our results also demonstrated StratDef’s superiority over the other MTDs in defending against transferability attacks. Furthermore, we showed that MTDs could be fingerprinted and that reconnaissance of these defenses was possible with a set of introductory methods that could be used to improve future attacks. Underpinned by our experimental findings, we offered several key recommendations for developing effective future MTDs. Mainly, we recommended that MTDs that behave statically should be avoided, while MTDs that dynamically utilize diverse constituent models with effective movement should be favored. Another key recommendation was to couple MTDs with greater system awareness. This would allow for the detection of query attacks in particular, which had been especially successful in evading the MTDs and other defenses that they were compared with.

In Chapter 5, we therefore presented a novel stateful defense against query attacks in the malware detection domain called MalProtect. The work in this chapter was related to our research objectives OBJ4 and OBJ5. Based on our empirical findings in the prior chapters, it was evident that prediction models could benefit from the emplacement of a stateful defense to provide greater system awareness to protect them against query attacks by monitoring user queries. Therefore, we proposed a novel stateful defense, and for the first time, we conducted a comprehensive evaluation of several stateful defenses applied to the ML-based malware detection domain, where we found prior stateful defenses to be ineffective. With a black-box attacker scenario (which constitutes the weakest threat model), a 60+% evasion rate was achievable across Android and Windows malware datasets. Meanwhile, our stateful defense could reduce the evasion rate of attacks by 80+% under black-box and gray-box scenarios. As we hypothesized, prior stateful defenses, relying solely on similarity or out-of-distribution detection, offered meager protection in this domain due to its constraints and inherent differences. Our stateful defense, MalProtect, was also designed to produce interpretable decisions that would be understandable by system analysts. In turn, this allowed us to explore how MalProtect would cope against an adaptive attacker who had complete knowledge of it, including how much each threat indicator contributed to the decisions. We found that MalProtect remained largely immune from such attacks as well. Our evaluation additionally showed that MalProtect offered more reliable predictions for non-adversarial queries than prior stateful defenses across a number of metrics, including the false positive rate.

The findings stemming from our work on novel defenses are highly encouraging and have led to some exciting future research directions. In the next section, we discuss a selection of intriguing, future research directions.

6.2 Future Work

The research and contributions presented in this dissertation are the result of several previously untouched research directions that led to one another. Naturally, these have expanded into a wider set of possible research directions that we discuss next. Specifically, these are: 1) Increasing Synergy Between MTDs & Stateful Defenses; 2) Dealing with Concept Drift in Adversarial ML Defenses; 3) Using Cyber-threat Intelligence in Adversarial ML Defenses; and 4) Increasing Collaboration Between Researchers & Practitioners. We discuss each of these next:

Increasing Synergy Between MTDs & Stateful Defenses

In Chapters 3 and 4, we uncovered a new research direction to explore whether MTDs can be improved by combining them with stateful defenses to introduce greater system awareness. In Chapter 5, we then provided a comprehensive evaluation in which some MTDs (as prediction models) were combined with stateful defenses. We showed that there was indeed an improvement in adversarial robustness. However, it remains to be seen whether the movement mechanisms of MTDs can be further refined in a more cohesive and synergistic manner. In particular, it may be interesting to explore whether an MTD's movement strategy or mechanism can be adapted in real-time according to the threats detected by a stateful defense. That is, an MTD may *move*

according to the current perceived threats or system conditions, which could involve modifying the MTD's parameters, adjusting the movement strategy, recycling and deprecating its constituent models, or using a completely different MTD altogether based on the analysis by a stateful defense. Not only would this create an additional layer of movement that may reduce the effectiveness of attacks, but it would further increase the uncertainty and complexity for the attacker.

Dealing with Concept Drift in Adversarial ML Defenses

A grand challenge within ML-based malware detection is model sustainability. As malware evolves, it becomes difficult to generalize models to detect unseen behavior, making classifiers unsustainable [109, 41, 80]. Due to this *concept drift*, models may make poor decisions when facing the latest threats (but not always [66]). Prior work has suggested periodically retraining a model [135, 103], though this may reduce a model's learning ability. Thus, the constant evolution of malware makes it a moving target in its own right. To deal with this, MTDs could use a detection system for concept drift and retire vulnerable constituent models [109]. For a stateful defense such as MalProtect, the defender may need to regularly evaluate the indicators with a detection framework to detect when modifications are necessary in order to offer effective performance.

Using Cyber-threat Intelligence in Adversarial ML Defenses

A key characteristic of malicious and adversarial queries is that they may reuse features or be developed in some systematic (albeit detectable) manner to evade defenses [197, 238]. Hence, an interesting research direction is to explore how cyber-threat intelligence (CTI) [197, 238] could be incorporated in an adversarial ML defense system where data could be aggregated from multiple sources and shared between vendors to increase protection. This data could include information about the latest threats, tactics, and techniques used by attackers, as well as databases of adversarial examples themselves. Such information could be utilized in different ways by defense systems. For example, the movement strategies of strategic MTDs could be adapted to the current perceived level of threat according to the CTI indicators. Meanwhile, for a stateful defense such as MalProtect, certain threat indicators could be developed that analyze queries using such additional CTI data for detecting attacks.

Increasing Collaboration Between Researchers & Practitioners

Increasing the collaboration between researchers and security practitioners remains an essential and open direction of work. Recent studies have exposed the lack of collaboration between academia and industry in adversarial machine learning research [17, 205, 91, 27, 34, 115]. Industry practitioners tend to give ML security a lower priority, sometimes owing to economic and business pressures or due to the perceived impracticality of attacks. Conversely, academic research has often overlooked the practicality of work when applied to real-world systems. For example, recent work has shown that the majority of adversarial attacks do not consider the fact that ML systems may be part of a wider pipeline, such as those that include pre-processors [199, 101], where an adversarial example may not even reach [81]. Therefore, more work needs to be done to foster greater collaboration between researchers and security practitioners as well as to raise awareness of the serious issues regarding the security of practical ML-based systems.

REFERENCES

- [1] Improving malicious document detection in gmail with deep learning, Feb 2020. URL <https://security.googleblog.com/2020/02/improving-malicious-document-detection.html>.
- [2] Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL <https://www.tensorflow.org/>.
- [3] Zainab Abaid, Mohamed Ali Kaafar, and Sanjay Jha. Quantifying the impact of adversarial evasion attacks on machine learning based android malware classifiers. In *2017 IEEE 16th International Symposium on Network Computing and Applications (NCA)*, pages 1–10, 2017. doi: 10.1109/NCA.2017.8171381.
- [4] Faitouri A Aboaoja, Anazida Zainal, Fuad A Ghaleb, Bander Ali Saleh Al-rimy, Taiseer Abdalla Elfadil Eisa, and Asma Abbas Hassan Elnour. Malware detection issues, challenges, and future directions: A survey. *Applied Sciences*, 12(17):8482, 2022.
- [5] Farhan Ahmed, Pratik Vaishnavi, Kevin Eykholt, and Amir Rahmati. Ares: A system-oriented wargame framework for adversarial ml. In *2022 IEEE Security and Privacy Workshops (SPW)*, pages 73–79, 2022. doi: 10.1109/SPW54247.2022.9833895.
- [6] Noor O. Ahmed and Bharat Bhargava. Mayflies: A moving target defense framework for distributed systems. In *Proceedings of the 2016 ACM Workshop on Moving Target Defense*, MTD ’16, page 59–64, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450345705. doi: 10.1145/2995272.2995283. URL <https://doi.org/10.1145/2995272.2995283>.
- [7] Mohammed Al-Asli and Taher Ahmed Ghaleb. Review of signature-based techniques in antivirus products. In *2019 International Conference on Computer and Information Sciences (ICCIS)*, pages 1–6, 2019. doi: 10.1109/ICCISci.2019.8716381.
- [8] Abdullah Al-Dujaili, Alex Huang, Erik Hemberg, and Una-May O’Reilly. Adversarial deep learning for robust detection of binary encoded malware. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 76–82. IEEE, 2018.
- [9] Huda Ali Alatwi and Charles Morisset. Adversarial machine learning in network intrusion detection domain: A systematic review. *arXiv preprint arXiv:2112.03315*, 2021.
- [10] Kevin Allix, Tegawendé F Bissyandé, Quentin Jérôme, Jacques Klein, Yves Le Traon, et al. Empirical assessment of machine learning-based malware detectors for android. *Empirical Software Engineering*, 21(1):183–211, 2016.
- [11] Kevin Allix, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. Androzoo: Collecting millions of android apps for the research community. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR ’16, pages 468–471, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4186-8. doi: 10.1145/2901739.2903508. URL <http://doi.acm.org/10.1145/2901739.2903508>.

- [12] Inc. Amazon Web Services. Machine learning on aws. URL <https://aws.amazon.com/machine-learning/>.
- [13] Abderrahmen Amich and Birhanu Eshete. Morphence: Moving target defense against adversarial examples. In *Annual Computer Security Applications Conference, ACSAC*, page 61–75, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450385794. doi: 10.1145/3485832.3485899. URL <https://doi.org/10.1145/3485832.3485899>.
- [14] Giuseppina Andresini, Feargus Pendlebury, Fabio Pierazzi, Corrado Loglisci, Annalisa Appice, and Lorenzo Cavallaro. Insomnia: Towards concept-drift robustness in network intrusion detection. In *Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security, AISEC '21*, page 111–122, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450386579. doi: 10.1145/3474369.3486864. URL <https://doi.org/10.1145/3474369.3486864>.
- [15] Giovanni Apruzzese, Michele Colajanni, Luca Ferretti, Alessandro Guido, and Mirco Marchetti. On the effectiveness of machine and deep learning for cyber security. In *2018 10th International Conference on Cyber Conflict (CyCon)*, pages 371–390, 2018. doi: 10.23919/CYCON.2018.8405026.
- [16] Giovanni Apruzzese, Pavel Laskov, Edgardo Montes de Oca, Wissam Mallouli, Luis Búrdalo Rapa, Athanasios Vasileios Grammatopoulos, and Fabio Di Franco. The role of machine learning in cybersecurity. *Digital Threats*, jul 2022. ISSN 2692-1626. doi: 10.1145/3545574. URL <https://doi.org/10.1145/3545574>. Just Accepted.
- [17] Giovanni Apruzzese, Hyrum S. Anderson, Savino Dambra, David Freeman, Fabio Pierazzi, and Kevin A. Roundy. "real attackers don't compute gradients": Bridging the gap between adversarial ml research and practice. In *Proceedings of the 1st IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, 2023.
- [18] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. Drebin: Effective and explainable detection of android malware in your pocket. In *Ndss*, volume 14, pages 23–26, 2014.
- [19] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. Dos and don'ts of machine learning in computer security. In *Proc. of the USENIX Security Symposium*, 2022.
- [20] Omer Aslan Aslan and Refik Samet. A comprehensive review on malware detection approaches. *IEEE Access*, 8:6249–6271, 2020. doi: 10.1109/ACCESS.2019.2963724.
- [21] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *International conference on machine learning*, pages 274–283. PMLR, 2018.
- [22] Buse Gul Atli, Sebastian Szyller, Mika Juuti, Samuel Marchal, and N Asokan. Extraction of complex dnn models: Real threat or boogeyman? In *International Workshop on Engineering Dependable and Secure Machine Learning Systems*, pages 42–57. Springer, 2020.
- [23] Federico Barbero, Feargus Pendlebury, Fabio Pierazzi, and Lorenzo Cavallaro. Transcending transcend: Revisiting malware classification in the presence of concept drift. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 805–823. IEEE, 2022.
- [24] Alexandru G. Bardas, Sathya Chandran Sundaramurthy, Xinming Ou, and Scott A. DeLoach. Mtd cbits: Moving target defense for cloud-based it systems. In *ESORICS*, 2017.

- [25] Peter L. Bartlett and Marten H. Wegkamp. Classification with a reject option using a hinge loss. *Journal of Machine Learning Research*, 9(59):1823–1840, 2008. URL <http://jmlr.org/papers/v9/bartlett08a.html>.
- [26] Arjun Nitin Bhagoji, Warren He, Bo Li, and Dawn Song. Practical black-box attacks on deep neural networks using efficient query mechanisms. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.
- [27] Lukas Bieringer, Kathrin Grosse, Michael Backes, Battista Biggio, and Katharina Krombholz. Industrial practitioners’ mental models of adversarial machine learning. In *Eighteenth Symposium on Usable Privacy and Security (SOUPS 2022)*, pages 97–116, Boston, MA, August 2022. USENIX Association. ISBN 978-1-939133-30-4. URL <https://www.usenix.org/conference/soups2022/presentation/bieringer>.
- [28] Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84:317–331, 2018.
- [29] Battista Biggio, Giorgio Fumera, and Fabio Roli. Adversarial pattern classification using multiple classifiers and randomisation. In *Proceedings of the 2008 Joint LAPR International Workshop on Structural, Syntactic, and Statistical Pattern Recognition*, SSPR & SPR ’08, page 500–509, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 9783540896883. doi: 10.1007/978-3-540-89689-0_54. URL https://doi.org/10.1007/978-3-540-89689-0_54.
- [30] Battista Biggio, Giorgio Fumera, and Fabio Roli. Multiple classifier systems for robust classifier design in adversarial environments. *International Journal of Machine Learning and Cybernetics*, 1:27–41, 2010.
- [31] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. *arXiv preprint arXiv:1206.6389*, 2012.
- [32] Battista Biggio, Iginio Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 387–402. Springer, 2013.
- [33] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.
- [34] Franziska Boenisch, Verena Battis, Nicolas Buchmann, and Maija Poikela. “i never thought about securing my machine learning systems”: A study of security and privacy awareness of machine learning practitioners. In *Proceedings of Mensch Und Computer 2021*, MuC ’21, page 520–546, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450386456. doi: 10.1145/3473856.3473869. URL <https://doi.org/10.1145/3473856.3473869>.
- [35] R. Braden, D. Clark, S. Crocker, and C. Huitema. Rfc1636: Report of iab workshop on security in the internet architecture - february 8-10, 1994, 1994.
- [36] Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. *arXiv preprint arXiv:1712.04248*, 2017.
- [37] Miles Brundage, Shahar Avin, Jack Clark, Helen Toner, Peter Eckersley, Ben Garfinkel, Allan Dafoe, Paul Scharre, Thomas Zeitzoff, Bobby Filar, et al. The malicious use of artificial intelligence: Forecasting, prevention, and mitigation. *arXiv preprint arXiv:1802.07228*, 2018.
- [38] Cristian Buciluă, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–541, 2006.

- [39] Saikiran Bulusu, Bhavya Kailkhura, Bo Li, Pramod K. Varshney, and Dawn Song. Anomalous example detection in deep learning: A survey. *IEEE Access*, 8:132330–132347, 2020. doi: 10.1109/ACCESS.2020.3010274.
- [40] Sebastian Buschjager, Kuan-Hsun Chen, Jian-Jia Chen, and Katharina Morik. Realization of random forest for real-time evaluation through tree framing. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 19–28, 2018. doi: 10.1109/ICDM.2018.00017.
- [41] Haipeng Cai and John Jenkins. Towards sustainable android malware detection. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceedings, ICSE '18*, page 350–351, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450356633. doi: 10.1145/3183440.3195004. URL <https://doi.org/10.1145/3183440.3195004>.
- [42] N. Carlini. A complete list of all (arxiv) adversarial example papers. URL <https://nicholas.carlini.com/writing/2019/all-adversarial-example-papers.html>.
- [43] Nicholas Carlini and David Wagner. Defensive distillation is not robust to adversarial examples. *arXiv preprint arXiv:1607.04311*, 2016.
- [44] Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM workshop on artificial intelligence and security*, pages 3–14, 2017.
- [45] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57. IEEE, 2017.
- [46] Nicholas Carlini, Anish Athalye, Nicolas Papernot, Wieland Brendel, Jonas Rauber, Dimitris Tsipras, Ian Goodfellow, Aleksander Madry, and Alexey Kurakin. On evaluating adversarial robustness. *arXiv preprint arXiv:1902.06705*, 2019.
- [47] Curtis Carmony, Xunchao Hu, Heng Yin, Abhishek Vasisht Bhaskar, and Mu Zhang. Extract me if you can: Abusing pdf parsers in malware detectors. In *Network and Distributed System Security Symposium*, 2016.
- [48] Anirban Chakraborty, Manaar Alam, Vishal Dey, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. Adversarial attacks and defences: A survey. *arXiv preprint arXiv:1810.00069*, 2018.
- [49] Chung-Kuan Chen, Si-Chen Lin, Szu-Chun Huang, Yung-Tien Chu, Chin-Laung Lei, and Chun-Ying Huang. Building machine learning-based threat hunting system from scratch. *Digital Threats*, 3(3), sep 2022. ISSN 2692-1626. doi: 10.1145/3491260. URL <https://doi.org/10.1145/3491260>.
- [50] Jianbo Chen, Michael I Jordan, and Martin J Wainwright. Hopskipjumpattack: A query-efficient decision-based attack. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1277–1294. IEEE, 2020.
- [51] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM workshop on artificial intelligence and security*, pages 15–26, 2017.
- [52] Steven Chen, Nicholas Carlini, and David Wagner. Stateful detection of black-box adversarial attacks. In *Proceedings of the 1st ACM Workshop on Security and Privacy on Artificial Intelligence*, pages 30–39, 2020.
- [53] Xiao Chen, Chaoran Li, Derui Wang, Sheng Wen, Jun Zhang, Surya Nepal, Yang Xiang, and Kui Ren. Android hiv: A study of repackaging malware for evading machine-learning detection. *IEEE Transactions on Information Forensics and Security*, 15:987–1001, 2019.

- [54] Yizheng Chen, Shiqi Wang, Dongdong She, and Suman Jana. On training robust {PDF} malware classifiers. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2343–2360, 2020.
- [55] Minhao Cheng, Cho-Jui Hsieh, Inderjit Dhillon, et al. Voting based ensemble improves robustness of defensive models. *arXiv preprint arXiv:2011.14031*, 2020.
- [56] Shuyu Cheng, Yinpeng Dong, Tianyu Pang, Hang Su, and Jun Zhu. Improving black-box adversarial attacks with a transfer-based prior. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL <https://proceedings.neurips.cc/paper/2019/file/32508f53f24c46f685870a075eaaa29c-Paper.pdf>.
- [57] Clarence Chio and David Freeman. *Machine learning and security: Protecting systems with data and algorithms*. " O'Reilly Media, Inc.", 2018.
- [58] Jin-Hee Cho, Dilli P Sharma, Hooman Alavizadeh, Seunghyun Yoon, Noam Ben-Asher, Terrence J Moore, Dong Seong Kim, Hyuk Lim, and Frederica F Nelson. Toward proactive, adaptive defense: A survey on moving target defense. *IEEE Communications Surveys & Tutorials*, 22(1):709–745, 2020.
- [59] François Chollet et al. Keras. <https://keras.io>, 2015.
- [60] Mihai Christodorescu, Somesh Jha, and Christopher Kruegel. Mining specifications of malicious behavior. In *Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ESEC-FSE '07, page 5–14, New York, NY, USA, 2007. Association for Computing Machinery. ISBN 9781595938114. doi: 10.1145/1287624.1287628. URL <https://doi.org/10.1145/1287624.1287628>.
- [61] Google Cloud. Google cloud. URL <https://cloud.google.com/>.
- [62] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *J. Mach. Learn. Res.*, 7:551–585, dec 2006. ISSN 1532-4435.
- [63] Emmanuel Gbenga Dada, Joseph Stephen Bassi, Haruna Chiroma, Shafi'i Muhammad Abdulhamid, Adebayo Olusola Adetunmbi, and Opeyemi Emmanuel Ajibuwa. Machine learning for email spam filtering: review, approaches and open research problems. *Heliyon*, 5(6):e01802, 2019. ISSN 2405-8440. doi: <https://doi.org/10.1016/j.heliyon.2019.e01802>. URL <https://www.sciencedirect.com/science/article/pii/S2405844018353404>.
- [64] Asim Darwaish, Farid Naït-Abdesselam, Chafiq Titouna, and Sumera Sattar. Robustness of image-based android malware detection under adversarial attacks. In *ICC 2021 - IEEE International Conference on Communications*, pages 1–6, 2021. doi: 10.1109/ICC42927.2021.9500425.
- [65] Prithviraj Dasgupta and Joseph Collins. A survey of game theoretic approaches for adversarial machine learning in cybersecurity tasks. *AI Magazine*, 40(2):31–43, 2019.
- [66] Dauodi et al. A deep dive inside drebin: An explorative analysis beyond android malware detection scores. *ACM Trans. Priv. Secur.*, 25(2), may 2022. ISSN 2471-2566. doi: 10.1145/3503463. URL <https://doi.org/10.1145/3503463>.
- [67] Luca Demetrio, Battista Biggio, Giovanni Lagorio, Fabio Roli, and Alessandro Armando. Explaining vulnerabilities of deep learning to adversarial malware binaries. *arXiv preprint arXiv:1901.03583*, 2019.
- [68] Luca Demetrio, Battista Biggio, Giovanni Lagorio, Fabio Roli, and Alessandro Armando. Functionality-preserving black-box optimization of adversarial windows malware. *IEEE Transactions on Information Forensics and Security*, 16:3469–3478, 2021.

- [69] 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. *IEEE Transactions on Dependable and Secure Computing*, 2017.
- [70] Jasjeet Dhaliwal and Saurabh Shintre. Gradient similarity: An explainable approach to detect adversarial attacks against deep learning. *arXiv preprint arXiv:1806.10707*, 2018.
- [71] John R. Douceur. The sybil attack. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, page 251–260, Berlin, Heidelberg, 2002. Springer-Verlag. ISBN 3540441794.
- [72] Stephan Dreiseitl and Lucila Ohno-Machado. Logistic regression and artificial neural network classification models: a methodology review. *Journal of biomedical informatics*, 35(5-6):352–359, 2002.
- [73] Yingjun Du, Haoliang Sun, Xiantong Zhen, Jun Xu, Yilong Yin, Ling Shao, and Cees GM Snoek. Metakernel: Learning variational random features with limited labels. *arXiv preprint arXiv:2105.03781*, 2021.
- [74] Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. HotFlip: White-box adversarial examples for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 31–36, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-2006. URL <https://aclanthology.org/P18-2006>.
- [75] Mohammadreza Ebrahimi, Ning Zhang, James Hu, Muhammad Taqi Raza, and Hsinchun Chen. Binary black-box evasion attacks against deep learning-based static malware detectors with adversarial byte-level language model. *arXiv preprint arXiv:2012.07994*, 2020.
- [76] Ivan Evtimov, Kevin Eykholt, Earlene Fernandes, Tadayoshi Kohno, Bo Li, Atul Prakash, Amir Rahmati, and Dawn Song. Robust physical-world attacks on machine learning models. *arXiv preprint arXiv:1707.08945*, 2(3):4, 2017.
- [77] Kevin Eykholt, Ivan Evtimov, Earlene 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 (CVPR)*, June 2018.
- [78] Ming Fan, Xiapu Luo, Jun Liu, Meng Wang, Chunyin Nong, Qinghua Zheng, and Ting Liu. Graph embedding based familial analysis of android malware using unsupervised learning. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 771–782, 2019. doi: 10.1109/ICSE.2019.00085.
- [79] Peter C. Fishburn. Letter to the editor—additive utilities with incomplete product sets: Application to priorities and assignments. *Operations Research*, 15(3):537–542, 1967. doi: 10.1287/opre.15.3.537. URL <https://doi.org/10.1287/opre.15.3.537>.
- [80] Xiaoqin Fu and Haipeng Cai. On the deterioration of learning-based malware detectors for android. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 272–273, 2019. doi: 10.1109/ICSE-Companion.2019.00110.
- [81] Joseph Gardiner and Shishir Nagaraja. On the security of machine learning in malware c&c detection: A survey. *ACM Comput. Surv.*, 49(3), dec 2016. ISSN 0360-0300. doi: 10.1145/3003816. URL <https://doi.org/10.1145/3003816>.
- [82] Dragoş Gavriliuţ, Mihai Cimpoeşu, Dan Anton, and Liviu Ciortuz. Malware detection using machine learning. In *2009 International Multiconference on Computer Science and Information Technology*, pages 735–741. IEEE, 2009.

- [83] Amira Ghenai and Yelena Mejova. Catching zika fever: Application of crowdsourcing and machine learning for tracking health misinformation on twitter. *arXiv preprint arXiv:1707.03778*, 2017.
- [84] Daniel Gibert, Carles Mateu, and Jordi Planes. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *Journal of Network and Computer Applications*, 153:102526, 2020. ISSN 1084-8045. doi: <https://doi.org/10.1016/j.jnca.2019.102526>. URL <https://www.sciencedirect.com/science/article/pii/S1084804519303868>.
- [85] Dong Gong, Lingqiao Liu, Vuong Le, Budhaditya Saha, Moussa Reda Mansour, Svetha Venkatesh, and Anton van den Hengel. Memorizing normality to detect anomaly: Memory-augmented deep autoencoder for unsupervised anomaly detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1705–1714, 2019.
- [86] Ian Goodfellow. A research agenda: Dynamic models to defend against correlated attacks. *arXiv preprint arXiv:1903.06293*, 2019.
- [87] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.
- [88] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel. Adversarial perturbations against deep neural networks for malware classification. *arXiv preprint arXiv:1606.04435*, 2016.
- [89] Kathrin Grosse, Praveen Manoharan, Nicolas Papernot, Michael Backes, and Patrick McDaniel. On the (statistical) detection of adversarial examples. *arXiv preprint arXiv:1702.06280*, 2017.
- [90] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. Adversarial examples for malware detection. In *European symposium on research in computer security*, pages 62–79. Springer, 2017.
- [91] Kathrin Grosse, Lukas Bieringer, Tarek Richard Besold, Battista Biggio, and Katharina Krombholz. "why do so?"—a practical perspective on machine learning security. *arXiv preprint arXiv:2207.05164*, 2022.
- [92] Qianyu Guo, Sen Chen, Xiaofei Xie, Lei Ma, Qiang Hu, Hongtao Liu, Yang Liu, Jianjun Zhao, and Xiaohong Li. An empirical study towards characterizing deep learning development and deployment across different frameworks and platforms. In *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering, ASE '19*, page 810–822. IEEE Press, 2019. ISBN 9781728125084. doi: 10.1109/ASE.2019.00080. URL <https://doi.org/10.1109/ASE.2019.00080>.
- [93] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2022. URL <https://www.gurobi.com>.
- [94] David J. Hand. Measuring classifier performance: A coherent alternative to the area under the roc curve. *Mach. Learn.*, 77(1):103–123, oct 2009. ISSN 0885-6125. doi: 10.1007/s10994-009-5119-5. URL <https://doi.org/10.1007/s10994-009-5119-5>.
- [95] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.
- [96] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [97] Warren He, James Wei, Xinyun Chen, Nicholas Carlini, and Dawn Song. Adversarial example defense: Ensembles of weak defenses are not strong. In *11th {USENIX} Workshop on Offensive Technologies ({WOOT} 17)*, 2017.

- [98] JB Heaton, Nicholas G Polson, and Jan Hendrik Witte. Deep learning in finance. *arXiv preprint arXiv:1602.06561*, 2016.
- [99] Dan Hendrycks, Nicholas Carlini, John Schulman, and Jacob Steinhardt. Unsolved problems in ml safety. *arXiv preprint arXiv:2109.13916*, 2021.
- [100] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015.
- [101] Sanghyun Hong, Nicholas Carlini, and Alexey Kurakin. Publishing efficient on-device models increases adversarial vulnerability. *arXiv preprint arXiv:2212.13700*, 2022.
- [102] Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. Black-box adversarial attacks with limited queries and information. In *International Conference on Machine Learning*, pages 2137–2146. PMLR, 2018.
- [103] Imam et al. A survey of attacks against twitter spam detectors in an adversarial environment. *Robotics*, 8(3), 2019. ISSN 2218-6581. doi: 10.3390/robotics8030050. URL <https://www.mdpi.com/2218-6581/8/3/50>.
- [104] Izmailov et al. *Combinatorial Boosting of Classifiers for Moving Target Defense Against Adversarial Evasion Attacks*, page 13–21. Association for Computing Machinery, New York, NY, USA, 2021. ISBN 9781450386586. URL <https://doi.org/10.1145/3474370.3485661>.
- [105] Todd Jackson, Babak Salamat, Gregor Wagner, Christian Wimmer, and Michael Franz. On the effectiveness of multi-variant program execution for vulnerability detection and prevention. In *Proceedings of the 6th International Workshop on Security Measurements and Metrics*, pages 1–8, 2010.
- [106] Todd Jackson, Christian Wimmer, and Michael Franz. Multi-variant program execution for vulnerability detection and analysis. In *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*, pages 1–4, 2010.
- [107] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Comput. Surv.*, 31(3): 264–323, sep 1999. ISSN 0360-0300. doi: 10.1145/331499.331504. URL <https://doi.org/10.1145/331499.331504>.
- [108] Sushil Jajodia, Anup K. Ghosh, V. S. Subrahmanian, Vipin Swarup, Cliff Wang, and Xiaoyang Sean Wang, editors. *Moving Target Defense II - Application of Game Theory and Adversarial Modeling*, volume 100 of *Advances in Information Security*. Springer, 2013. ISBN 978-1-4614-5415-1. doi: 10.1007/978-1-4614-5416-8. URL <https://doi.org/10.1007/978-1-4614-5416-8>.
- [109] Jordaney et al. Transcend: Detecting concept drift in malware classification models. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 625–642, Vancouver, BC, August 2017. USENIX Association. ISBN 978-1-931971-40-9. URL <https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/jordaney>.
- [110] Mika Juuti, Sebastian Szyller, Samuel Marchal, and N Asokan. Prada: protecting against dnn model stealing attacks. In *2019 IEEE European Symposium on Security and Privacy (EuroSecP)*, pages 512–527. IEEE, 2019.
- [111] Sanjay Kariyappa and Moinuddin K Qureshi. Improving adversarial robustness of ensembles with diversity training. *arXiv preprint arXiv:1901.09981*, 2019.
- [112] Sanjay Kariyappa and Moinuddin K. Qureshi. Defending against model stealing attacks with adaptive misinformation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

- [113] A. Kerckhoffs. La cryptographie militaire. *Journal des Sciences Militaires*, pages 161–191, 1883.
- [114] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL <https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf>.
- [115] Ram Shankar Siva Kumar, Magnus Nyström, John Lambert, Andrew Marshall, Mario Goertzel, Andi Comissioneru, Matt Swann, and Sharon Xia. Adversarial machine learning-industry perspectives. In *2020 IEEE Security and Privacy Workshops (SPW)*, pages 69–75. IEEE, 2020.
- [116] Alexey Kurakin, Ian Goodfellow, Samy Bengio, et al. Adversarial examples in the physical world, 2016.
- [117] Raphael Labaca-Castro, Luis Muñoz-González, Feargus Pendlebury, Gabi Dreo Rodosek, Fabio Pierazzi, and Lorenzo Cavallaro. Universal adversarial perturbations for malware. *arXiv preprint arXiv:2102.06747*, 2021.
- [118] Pavel Laskov et al. Practical evasion of a learning-based classifier: A case study. In *2014 IEEE symposium on security and privacy*, pages 197–211. IEEE, 2014.
- [119] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.
- [120] Aoxue Li, Tiange Luo, Tao Xiang, Weiran Huang, and Liwei Wang. Few-shot learning with global class representations. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9715–9724, 2019.
- [121] Chaoran Li, Xiao Chen, Derui Wang, Sheng Wen, Muhammad Ejaz Ahmed, Seyit Camtepe, and Yang Xiang. Backdoor attack on machine learning based android malware detectors. *IEEE Transactions on Dependable and Secure Computing*, 19(5):3357–3370, 2022. doi: 10.1109/TDSC.2021.3094824.
- [122] Deqiang Li and Qianmu Li. Adversarial deep ensemble: Evasion attacks and defenses for malware detection. *IEEE Transactions on Information Forensics and Security*, 15:3886–3900, 2020.
- [123] Deqiang Li, Qianmu Li, Yanfang Ye, and Shouhuai Xu. Enhancing deep neural networks against adversarial malware examples. *arXiv preprint arXiv:2004.07919*, 2020.
- [124] Deqiang Li, Qianmu Li, Yanfang Ye, and Shouhuai Xu. A framework for enhancing deep neural networks against adversarial malware. *IEEE Transactions on Network Science and Engineering*, 8(1): 736–750, 2021.
- [125] Deqiang Li, Qianmu Li, Yanfang (Fanny) Ye, and Shouhuai Xu. Arms race in adversarial malware detection: A survey. *ACM Comput. Surv.*, 55(1), nov 2021. ISSN 0360-0300. doi: 10.1145/3484491. URL <https://doi.org/10.1145/3484491>.
- [126] 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*, pages 596–608, 2021.
- [127] Heng Li, ShiYao Zhou, Wei Yuan, Jiahuan Li, and Henry Leung. Adversarial-example attacks toward android malware detection system. *IEEE Systems Journal*, 14(1):653–656, 2020. doi: 10.1109/JSYST.2019.2906120.
- [128] Huichen Li, Xiaojun Xu, Xiaolu Zhang, Shuang Yang, and Bo Li. Qeba: Query-efficient boundary-based blackbox attack. In *CVPR*, 2020.

- [129] Huichen Li, Xiaojun Xu, Xiaolu Zhang, Shuang Yang, and Bo Li. Qeba: Query-efficient boundary-based blackbox attack. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1221–1230, 2020.
- [130] Huiying Li, Shawn Shan, Emily Wenger, Jiayun Zhang, Haitao Zheng, and Ben Y. Zhao. Blacklight: Scalable defense for neural networks against Query-Based Black-Box attacks. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 2117–2134, Boston, MA, August 2022. USENIX Association. ISBN 978-1-939133-31-1. URL <https://www.usenix.org/conference/usenixsecurity22/presentation/li-huiying>.
- [131] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. Delving into transferable adversarial examples and black-box attacks. *arXiv preprint arXiv:1611.02770*, 2016.
- [132] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.
- [133] Jiawei Ma, Hanchen Xie, Guangxing Han, Shih-Fu Chang, Aram Galstyan, and Wael Abd-Almageed. Partner-assisted learning for few-shot image classification. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10573–10582, 2021.
- [134] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.
- [135] Federico Maggi, William Robertson, Christopher Kruegel, and Giovanni Vigna. Protecting a moving target: Addressing web application concept drift. In *Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection, RAID '09*, page 21–40, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 9783642043413. doi: 10.1007/978-3-642-04342-0_2. URL https://doi.org/10.1007/978-3-642-04342-0_2.
- [136] Mohammad Maghsoudimehrabani, Amin Azmoodeh, Ali Dehghantanha, Behrouz Zolfaghari, and Gautam Srivastava. Proactive detection of query-based adversarial scenarios in nlp systems. In *Proceedings of the 15th ACM Workshop on Artificial Intelligence and Security, AISec'22*, page 103–113, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450398800. doi: 10.1145/3560830.3563727. URL <https://doi.org/10.1145/3560830.3563727>.
- [137] Peter Martin, Jian Fan, Taejin Kim, Konrad Vesey, and Lloyd Greenwald. Toward effective moving target defense against adversarial ai. In *MILCOM 2021 - 2021 IEEE Military Communications Conference (MILCOM)*, pages 993–998, 2021. doi: 10.1109/MILCOM52596.2021.9652915.
- [138] Dongyu Meng and Hao Chen. Magnet: a two-pronged defense against adversarial examples. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pages 135–147, 2017.
- [139] Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. On detecting adversarial perturbations. *arXiv preprint arXiv:1702.04267*, 2017.
- [140] Brad Miller, Alex Kantchelian, Michael Carl Tschantz, Sadia Afroz, Rekha Bachwani, Riyaz Faizullahoy, Ling Huang, Vaishaal Shankar, Tony Wu, George Yiu, et al. Reviewer integration and performance measurement for malware detection. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 122–141. Springer, 2016.
- [141] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. Kitsune: an ensemble of autoencoders for online network intrusion detection. *arXiv preprint arXiv:1802.09089*, 2018.

- [142] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2574–2582, 2016.
- [143] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1765–1773, 2017.
- [144] Stylianos Moschoglou, Athanasios Papaioannou, Christos Sagonas, Jiankang Deng, Irene Kotsia, and Stefanos Zafeiriou. Agedb: the first manually collected, in-the-wild age database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshop*, volume 2, page 5, 2017.
- [145] Andreas Moser, Christopher Kruegel, and Engin Kirda. Limits of static analysis for malware detection. In *23rd Annual Computer Security Applications Conference (ACSAC 2007), December 10-14, 2007, Miami Beach, Florida, USA*, pages 421–430. IEEE Computer Society, 2007. doi: 10.1109/ACSAC.2007.21. URL <https://doi.org/10.1109/ACSAC.2007.21>.
- [146] Andreas Moser, Christopher Kruegel, and Engin Kirda. Limits of static analysis for malware detection. In *Twenty-third annual computer security applications conference (ACSAC 2007)*, pages 421–430. IEEE, 2007.
- [147] Azqa Nadeem, Daniël Vos, Clinton Cao, Luca Pajola, Simon Dieck, Robert Baumgartner, and Sicco Verwer. Sok: Explainable machine learning for computer security applications. *arXiv preprint arXiv:2208.10605*, 2022.
- [148] Milad Nasr, Alireza Bahramali, and Amir Houmansadr. Defeating DNN-Based traffic analysis systems in Real-Time with blind adversarial perturbations. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2705–2722. USENIX Association, August 2021. ISBN 978-1-939133-24-3. URL <https://www.usenix.org/conference/usenixsecurity21/presentation/nasr>.
- [149] Shirin Nilizadeh, François Labrèche, Alireza Sedighian, Ali Zand, José Fernandez, Christopher Kruegel, Gianluca Stringhini, and Giovanni Vigna. Poised: Spotting twitter spam off the beaten paths. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1159–1174, 2017.
- [150] Department of Homeland Security. Moving target defense. URL <https://www.dhs.gov/science-and-technology/csd-mtd>.
- [151] Philip O’Kane, Sakir Sezer, and Kieran McLaughlin. Obfuscation: The hidden malware. *IEEE Security & Privacy*, 9(5):41–47, 2011. doi: 10.1109/MSP.2011.98.
- [152] Hamed Okhravi, Thomas Hobson, David Bigelow, and William Streilein. Finding focus in the blur of moving-target techniques. *IEEE Security & Privacy*, 12(2):16–26, 2014. doi: 10.1109/MSP.2013.137.
- [153] Lucky Onwuzurike, Enrico Mariconti, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon Ross, and Gianluca Stringhini. Mamadroid: Detecting android malware by building markov chains of behavioral models (extended version). *ACM Transactions on Privacy and Security (TOPS)*, 22(2):1–34, 2019.
- [154] Daniel W Otter, Julian R Medina, and Jugal K Kalita. A survey of the usages of deep learning for natural language processing. *IEEE transactions on neural networks and learning systems*, 32(2):604–624, 2020.
- [155] Soham Pal, Yash Gupta, Aditya Kanade, and Shirish Shevade. Stateful detection of model extraction attacks. *arXiv preprint arXiv:2107.05166*, 2021.

- [156] Guansong Pang, Chunhua Shen, Longbing Cao, and Anton Van Den Hengel. Deep learning for anomaly detection: A review. *ACM Comput. Surv.*, 54(2), mar 2021. ISSN 0360-0300. doi: 10.1145/3439950. URL <https://doi.org/10.1145/3439950>.
- [157] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z.B. Celik, and A. Swami. Practical black-box attacks against machine learning. *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pages 506–519, 2018.
- [158] Nicolas Papernot and Patrick McDaniel. On the effectiveness of defensive distillation. *arXiv preprint arXiv:1607.05113*, 2016.
- [159] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv preprint arXiv:1605.07277*, 2016.
- [160] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *2016 IEEE European symposium on security and privacy (EuroSecP)*, pages 372–387. IEEE, 2016.
- [161] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 582–597. IEEE, 2016.
- [162] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael P Wellman. Sok: Security and privacy in machine learning. In *2018 IEEE European Symposium on Security and Privacy (EuroSecP)*, pages 399–414. IEEE, 2018.
- [163] Aswin Chidambaram Pappa, Aditya Ashok, and Manimaran Govindarasu. Moving target defense for securing smart grid communications: Architecture, implementation & evaluation. In *2017 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, pages 1–5, 2017. doi: 10.1109/ISGT.2017.8085954.
- [164] Praveen Paruchuri, Jonathan P Pearce, Janusz Marecki, Milind Tambe, Fernando Ordonez, and Sarit Kraus. Playing games for security: An efficient exact algorithm for solving bayesian stackelberg games. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, pages 895–902. International Foundation for Autonomous Agents and Multiagent Systems, 2008.
- [165] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830, 2011.
- [166] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. Tesseract: Eliminating experimental bias in malware classification across space and time. In *Proceedings of the 28th USENIX Conference on Security Symposium, SEC’19*, page 729–746, USA, 2019. USENIX Association. ISBN 9781939133069.
- [167] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. {TESSERACT}: Eliminating experimental bias in malware classification across space and time. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 729–746, 2019.
- [168] Pierazzi et al. Intriguing properties of adversarial ml attacks in the problem space. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1308–1325. IEEE Computer Society, 2020. doi: 10.1109/SP40000.2020.00073. URL <https://doi.ieeecomputersociety.org/10.1109/SP40000.2020.00073>.

- [169] R. Podschwadt and H. Takabi. On Effectiveness of Adversarial Examples and Defenses for Malware Classification. *International Conference on Security and Privacy in Communication Systems*, pages 380–393, 2019.
- [170] Friedrich Pukelsheim. The three sigma rule. *The American Statistician*, 48(2):88–91, 1994. ISSN 00031305. URL <http://www.jstor.org/stable/2684253>.
- [171] Qian et al. Ei-mtd: Moving target defense for edge intelligence against adversarial attacks. *ACM Trans. Priv. Secur.*, 25(3), may 2022. ISSN 2471-2566. doi: 10.1145/3517806. URL <https://doi.org/10.1145/3517806>.
- [172] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles K Nicholas. Malware detection by eating a whole exe. In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- [173] Edward Raff, Richard Zak, Gary Lopez Munoz, William Fleming, Hyrum S Anderson, Bobby Filar, Charles Nicholas, and James Holt. Automatic yara rule generation using biclustering. In *Proceedings of the 13th ACM Workshop on Artificial Intelligence and Security*, pages 71–82, 2020.
- [174] Aqib Rashid and Jose Such. Stratdef: Strategic defense against adversarial attacks in ml-based malware detection. *Computers & Security*, 2023.
- [175] Aqib Rashid and Jose Such. Effectiveness of moving target defenses for adversarial attacks in ml-based malware detection. *arXiv preprint arXiv:2302.00537*, 2023.
- [176] Aqib Rashid and Jose Such. Malprotect: Stateful defense against adversarial query attacks in ml-based malware detection. *IEEE Transactions on Information Forensics and Security*, pages 1–1, 2023. doi: 10.1109/TIFS.2023.3293959.
- [177] Hemant Rathore, Sanjay K Sahay, Piyush Nikam, and Mohit Sewak. Robust android malware detection system against adversarial attacks using q-learning. *Information Systems Frontiers*, 23(4):867–882, 2021.
- [178] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *International conference on learning representations*, 2016.
- [179] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Brij B. Gupta, Xiaojiang Chen, and Xin Wang. A survey of deep active learning. *ACM Comput. Surv.*, 54(9), oct 2021. ISSN 0360-0300. doi: 10.1145/3472291. URL <https://doi.org/10.1145/3472291>.
- [180] Leslie Rice, Eric Wong, and Zico Kolter. Overfitting in adversarially robust deep learning. In *International Conference on Machine Learning*, pages 8093–8104. PMLR, 2020.
- [181] Ishai Rosenberg, Asaf Shabtai, Lior Rokach, and Yuval Elovici. Generic black-box end-to-end attack against state of the art api call based malware classifiers. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 490–510. Springer, 2018.
- [182] Ishai Rosenberg, Asaf Shabtai, Yuval Elovici, and Lior Rokach. Query-efficient black-box attack against sequence-based malware classifiers. In *Annual Computer Security Applications Conference*, pages 611–626, 2020.
- [183] Christian Rossow, Christian J. Dietrich, Chris Grier, Christian Kreibich, Vern Paxson, Norbert Pohlmann, Herbert Bos, and Maarten van Steen. Prudent practices for designing malware experiments: Status quo and outlook. In *2012 IEEE Symposium on Security and Privacy*, pages 65–79, 2012. doi: 10.1109/SP.2012.14.

- [184] Roy et al. A moving target defense against adversarial machine learning. In *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, SEC '19, page 383–388, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367332. doi: 10.1145/3318216.3363338. URL <https://doi.org/10.1145/3318216.3363338>.
- [185] Everton Jose Santana, Ricardo Petri Silva, Bruno Bogaz Zarpelão, and Sylvio Barbon Junior. Detecting and mitigating adversarial examples in regression tasks: A photovoltaic power generation forecasting case study. *Information*, 12(10):394, 2021.
- [186] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.
- [187] Lea Schönherr, Katharina Kohls, Steffen Zeiler, Thorsten Holz, and Dorothea Kolossa. Adversarial attacks against automatic speech recognition systems via psychoacoustic hiding. *arXiv preprint arXiv:1808.05665*, 2018.
- [188] Pascal Schöttle, Alexander Schlögl, Cecilia Pasquini, and Rainer Böhme. Detecting adversarial examples - a lesson from multimedia security. In *2018 26th European Signal Processing Conference (EUSIPCO)*, pages 947–951, 2018. doi: 10.23919/EUSIPCO.2018.8553164.
- [189] Daniel Scofield, Craig Miles, and Stephen Kuhn. Automated model learning for accurate detection of malicious digital documents. *Digital Threats*, 1(3), aug 2020. ISSN 2692-1626. doi: 10.1145/3379505. URL <https://doi.org/10.1145/3379505>.
- [190] Sailik Sengupta, Satya Gautam Vadlamudi, Subbarao Kambhampati, Adam Doupé, Ziming Zhao, Marthony Taguinod, and Gail-Joon Ahn. A game theoretic approach to strategy generation for moving target defense in web applications. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '17, page 178–186, Richland, SC, 2017. International Foundation for Autonomous Agents and Multiagent Systems.
- [191] Sailik Sengupta, Tathagata Chakraborti, and Subbarao Kambhampati. Mtdeep: boosting the security of deep neural nets against adversarial attacks with moving target defense. In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- [192] Giorgio Severi, Jim Meyer, Scott Coull, and Alina Oprea. Explanation-guided backdoor poisoning attacks against malware classifiers. In *30th {USENIX} Security Symposium ({USENIX} Security 21)*, 2021.
- [193] Mohit Sewak, Sanjay K Sahay, and Hemant Rathore. Adversarialuscat: An adversarial-drl based obfuscator and metamorphic malware swarm generator. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–9. IEEE, 2021.
- [194] Raja Khurram Shahzad and Niklas Lavesson. Comparative analysis of voting schemes for ensemble-based malware detection. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, 4(1):98–117, 2013.
- [195] Shawn Shan, Arjun Nitin Bhagoji, Haitao Zheng, and Ben Y. Zhao. Patch-based defenses against web fingerprinting attacks. In *Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security*, AISec '21, page 97–109, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450386579. doi: 10.1145/3474369.3486875. URL <https://doi.org/10.1145/3474369.3486875>.
- [196] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE symposium on security and privacy (SP)*, pages 3–18. IEEE, 2017.

- [197] Xiaokui Shu, Frederico Araujo, Douglas L Schales, Marc Ph Stoecklin, Jiyong Jang, Heqing Huang, and Josyula R Rao. Threat intelligence computing. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1883–1898, 2018.
- [198] Hispasec Sistemas. Virustotal. URL <https://www.virustotal.com/>.
- [199] Chawin Sitawarin, Florian Tramèr, and Nicholas Carlini. Preprocessors matter! realistic decision-based attacks on machine learning systems. *arXiv preprint arXiv:2210.03297*, 2022.
- [200] Qun Song, Zhenyu Yan, and Rui Tan. Deepmtt: Moving target defense for deep visual sensing against adversarial examples. *ACM Trans. Sen. Netw.*, 18(1), oct 2021. ISSN 1550-4859. doi: 10.1145/3469032. URL <https://doi.org/10.1145/3469032>.
- [201] Song et al. Mab-malware: A reinforcement learning framework for blackbox generation of adversarial malware. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS ’22, page 990–1003, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450391405. doi: 10.1145/3488932.3497768. URL <https://doi.org/10.1145/3488932.3497768>.
- [202] Jack W Stokes, De Wang, Mady Marinescu, Marc Marino, and Brian Bussone. Attack and defense of dynamic analysis-based, adversarial neural malware classification models. *arXiv preprint arXiv:1712.05919*, 2017.
- [203] Octavian Suci, Radu Marginean, Yigitcan Kaya, Hal Daume III, and Tudor Dumitras. When does machine learning FAIL? generalized transferability for evasion and poisoning attacks. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1299–1316, Baltimore, MD, August 2018. USENIX Association. ISBN 978-1-939133-04-5. URL <https://www.usenix.org/conference/usenixsecurity18/presentation/suci>.
- [204] Octavian Suci, Scott E Coull, and Jeffrey Johns. Exploring adversarial examples in malware detection. In *2019 IEEE Security and Privacy Workshops (SPW)*, pages 8–14. IEEE, 2019.
- [205] Zhichuang Sun, Ruimin Sun, Long Lu, and Alan Mislove. Mind your weight (s): A large-scale study on insufficient machine learning model protection in mobile apps. *arXiv preprint arXiv:2002.07687*, 2020.
- [206] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.
- [207] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2014.
- [208] Rahim Taheri, Reza Javidan, Mohammad Shojafar, Zahra Pooranian, Ali Miri, and Mauro Conti. On defending against label flipping attacks on malware detection systems. *Neural Computing and Applications*, 32(18):14781–14800, 2020.
- [209] Milind Tambe. *Security and game theory: algorithms, deployed systems, lessons learned*. Cambridge university press, 2011.
- [210] Cheng Tang, Damien Garreau, and Ulrike von Luxburg. When do random forests fail? In *NeurIPS*, pages 2987–2997, 2018.
- [211] Microsoft Defender Security Research Team. New machine learning model sifts through the good to unearth the bad in evasive malware, Aug 2019. URL <https://www.microsoft.com/en-us/security/blog/2019/07/25/new-machine-learning-model-sifts-through-the-good-to-unearth-the-bad-in-evasive-malware/>.

- [212] Romain Thomas. Lief - library to instrument executable formats. <https://lief.quarkslab.com/>, April 2017.
- [213] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction apis. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 601–618, 2016.
- [214] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*, 2017.
- [215] Florian Tramer, Nicholas Carlini, Wieland Brendel, and Aleksander Madry. On adaptive attacks to adversarial example defenses. *Advances in Neural Information Processing Systems*, 33:1633–1645, 2020.
- [216] Jonathan Uesato, Brendan O’donoghue, Pushmeet Kohli, and Aaron Oord. Adversarial risk and the dangers of evaluating against weak attacks. In *International Conference on Machine Learning*, pages 5025–5034. PMLR, 2018.
- [217] VenkataRamani Varanasi and Shaik Razia. Network intrusion detection using machine learning, deep learning - a review. In *2022 4th International Conference on Smart Systems and Inventive Technology (ICSSIT)*, pages 1618–1624, 2022. doi: 10.1109/ICSSIT53264.2022.9716469.
- [218] Sebastian Vollmer, Bilal A Mateen, Gergo Bohner, Franz J Király, Rayid Ghani, Pall Jonsson, Sarah Cumbers, Adrian Jonas, Katherine SL McAllister, Puja Myles, et al. Machine learning and ai research for patient benefit: 20 critical questions on transparency, replicability, ethics and effectiveness. *arXiv preprint arXiv:1812.10404*, 2018.
- [219] Binghui Wang and Neil Zhenqiang Gong. Stealing hyperparameters in machine learning. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 36–52. IEEE, 2018.
- [220] Huangxin Wang, Fei Li, and Songqing Chen. Towards cost-effective moving target defense against ddos and covert channel attacks. In *Proceedings of the 2016 ACM Workshop on Moving Target Defense, MTD ’16*, page 15–25, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450345705. doi: 10.1145/2995272.2995281. URL <https://doi.org/10.1145/2995272.2995281>.
- [221] Qinglong Wang, Wenbo Guo, Kaixuan Zhang, Xinyu Xing, C Lee Giles, and Xue Liu. Random feature nullification for adversary resistant deep architecture. *arXiv preprint arXiv:1610.01239*, 2016.
- [222] Weiwei Wang, Xinli Xiong, Songhe Wang, and Jingye Zhang. Mtdnnf: Building the security framework for deep neural network by moving target defense. In *2020 3rd International Conference on Algorithms, Computing and Artificial Intelligence*, pages 1–1, 2020.
- [223] Yizhen Wang, Mohannad Alhanahnah, Xiaozhu Meng, Ke Wang, Mihai Christodorescu, and Somesh Jha. Robust learning against relational adversaries. In *Advances in Neural Information Processing Systems*.
- [224] Bryan C Ward, Steven R Gomez, Richard Skowyra, David Bigelow, Jason Martin, James Landry, and Hamed Okhravi. Survey of cyber moving targets second edition. *MIT Lincoln Laboratory Lexington United States, Tech. Rep*, 2018.
- [225] Alexander Warnecke, Daniel Arp, Christian Wressnegger, and Konrad Rieck. Evaluating explanation methods for deep learning in security. In *2020 IEEE european symposium on security and privacy (EuroSecP)*, pages 158–174. IEEE, 2020.
- [226] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, and Alan Yuille. Mitigating adversarial effects through randomization. *arXiv preprint arXiv:1711.01991*, 2017.

- [227] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv preprint arXiv:1704.01155*, 2017.
- [228] Ziang Yan, Yiwen Guo, Jian Liang, and Changshui Zhang. Policy-driven attack: Learning to query for hard-label black-box adversarial examples. In *International Conference on Learning Representations*, 2021. URL <https://openreview.net/forum?id=pzpytjk3Xb2>.
- [229] 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 33rd Annual Computer Security Applications Conference*, pages 288–302, 2017.
- [230] Huaxiu Yao, Ying Wei, Junzhou Huang, and Zhenhui Li. Hierarchically structured meta-learning. In *International Conference on Machine Learning*, pages 7045–7054. PMLR, 2019.
- [231] Han-Jia Ye and Wei-Lun Chao. How to train your maml to excel in few-shot classification. *arXiv preprint arXiv:2106.16245*, 2021.
- [232] Suleiman Y Yerima and Sakir Sezer. Droidfusion: A novel multilevel classifier fusion approach for android malware detection. *IEEE transactions on cybernetics*, 49(2):453–466, 2018.
- [233] Honggang Yu, Kaichen Yang, Teng Zhang, Yun-Yun Tsai, Tsung-Yi Ho, and Yier Jin. Cloudleak: Large-scale deep learning models stealing through adversarial examples. In *NDSS*, 2020.
- [234] Zhanyuan Zhang, Yizheng Chen, and David Wagner. Seat: Similarity encoder by adversarial training for detecting model extraction attack queries. In *Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security*, AISEC ’21, page 37–48, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450386579. doi: 10.1145/3474369.3486863. URL <https://doi.org/10.1145/3474369.3486863>.
- [235] 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. CCS ’21, page 3218–3235, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450384544. doi: 10.1145/3460120.3485387. URL <https://doi.org/10.1145/3460120.3485387>.
- [236] Min Zheng, Mingshen Sun, and John C.S. Lui. Droid analytics: A signature based analytic system to collect, extract, analyze and associate android malware. In *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pages 163–171, 2013. doi: 10.1109/TrustCom.2013.25.
- [237] Xiaojin Jerry Zhu. Semi-supervised learning literature survey. 2005.
- [238] Ziyun Zhu and Tudor Dumitras. Chainsmith: Automatically learning the semantics of malicious campaigns by mining threat intelligence reports. In *2018 IEEE European Symposium on Security and Privacy (EuroSecP)*, pages 458–472. IEEE, 2018.
- [239] Rui Zhuang, Su Zhang, Alex Bardas, Scott A. DeLoach, Xinming Ou, and Anoop Singhal. Investigating the application of moving target defenses to network security. In *2013 6th International Symposium on Resilient Control Systems (ISRCS)*, pages 162–169, 2013. doi: 10.1109/ISRCS.2013.6623770.
- [240] Nedim žRndić and Pavel Laskov. Hidost: A static machine-learning-based detector of malicious files. *EURASIPJ. Inf. Secur.*, 2016(1), dec 2016. ISSN 1687-4161. doi: 10.1186/s13635-016-0045-0. URL <https://doi.org/10.1186/s13635-016-0045-0>.

APPENDIX A

PERMITTED PERTURBATIONS FOR ANDROID DATASETS

A.1 Permitted Perturbations for Android Datasets

DREBIN [18] and AndroZoo [11] are Android datasets, both of which can be divided into eight feature families comprised of extracted static features such as permissions, API calls, hardware requests, and URL requests. According to industry documentation and prior work (e.g., [124, 3, 123, 8, 168, 117, 174]), features may be added or removed during adversarial attacks to traverse the decision boundary, based on the feature family.

However, when generating adversarial examples in this domain, a core constraint is the preservation of malicious functionality (see Section 2.2.3). As we operate in the feature-space, we offer a lower bound of functionality preservation. For example, attacks cannot remove features from the manifest file nor intent filter, and component names must be consistently named. Therefore, the table below enumerates the perturbations for each feature family that are allowed. For example, if a feature belonging to the S2 family is removed by an attack, then its original value is restored as it is not permitted to be removed.

	Feature families	Addition	Removal
manifest	S1 Hardware	✓	✗
	S2 Requested permissions	✓	✗
	S3 Application components	✓	✓
	S4 Intents	✓	✗
dexcode	S5 Restricted API Calls	✓	✓
	S6 Used permission	✗	✗
	S7 Suspicious API calls	✓	✓
	S8 Network addresses	✓	✓

Table A.1: Permitted perturbations for Android datasets. These are determined by consulting industry documentation and prior work [124, 3, 123, 8, 168, 117, 174].

APPENDIX B

STRATEGIC DEFENSE AGAINST ADVERSARIAL ATTACKS

B.1 Examples of StratDef Strategies

5 attacker types, 2 datasets and 6 StratDef configurations, leads to 60 strategy vectors. For brevity, we only include some examples of the StratDef strategies for both datasets. For each strategy vector, the rows correspond to the models selected through our model selection methods (Best & Variety). Within each row, the probability of that model being selected at a particular attack intensity is listed.

	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
DT-AT-0.01	0	0	0	0	0	0	0	0	0	0	0
DT-AT-0.1	0	0	0	0	0	0.985507	0.985507	0.985507	0.985507	0.985507	0.985507
DT-RFN	0	0	0	0.510791	0.510791	0.014493	0.014493	0.014493	0.014493	0.014493	0.014493
NN-RFN	0	0	0	0	0	0	0	0	0	0	0
RF	0	0	0	0	0	0	0	0	0	0	0
RF-AT-0.25	1	1	1	0.489209	0.489209	0	0	0	0	0	0
RF-RFN	0	0	0	0	0	0	0	0	0	0	0
SECSVM-AT-0.25	0	0	0	0	0	0	0	0	0	0	0
SVM	0	0	0	0	0	0	0	0	0	0	0

Table B.1: DREBIN, StratDef-Best-GT, Strong attacker

	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
DT-AT-0.01	0	0	0	0	0	0	0	0	0	0	0
DT-AT-0.1	0	0	0.133333	0.133333	0.266667	0.266667	0.266667	0.266667	0.333333	0.333333	0.333333
DT-RFN	0	0	0	0	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667	0.2
NN-RFN	0	0.066667	0	0	0	0	0	0	0	0	0
RF	0	0.266667	0.2	0.266667	0.2	0.2	0.2	0.133333	0.2	0.133333	0.066667
RF-AT-0.25	1	0.333333	0.333333	0.333333	0.333333	0.333333	0.333333	0.333333	0.266667	0.266667	0.266667
RF-RFN	0	0.2	0.266667	0.2	0.133333	0.133333	0.133333	0.2	0.133333	0.2	0.133333
SECSVM-AT-0.25	0	0.133333	0.066667	0.066667	0	0	0	0	0	0	0
SVM	0	0	0	0	0	0	0	0	0	0	0

Table B.2: DREBIN, StratDef-Best-Ranked, Strong attacker

	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
DT_AT_0.01	0	0	0	0	0	0	0	0	0	0	0
DT_AT_0.1	0	0	0	0	0	0	0.890756	0.890756	0.890756	0.890756	0.897638
DT_AT_0.25	0	0	0	0	0	0	0	0	0	0	0
NN	0	0	0	0	0	0	0	0	0	0	0
NN_AT_0.25	0	0	0	0	0	0	0	0	0	0	0.102362
RF_AT_0.25	1	1	1	1	1	1	0.109244	0.109244	0.109244	0.109244	0
SECSVM_AT_0.05	0	0	0	0	0	0	0	0	0	0	0
SECSVM_AT_0.25	0	0	0	0	0	0	0	0	0	0	0
SVM	0	0	0	0	0	0	0	0	0	0	0

Table B.3: DREBIN, StratDef-Variety-GT, Weak attacker

	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
DT_AT_0.01	0	0	0	0	0	0	0	0	0	0	0
DT_AT_0.1	0	0.1	0.2	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3
NN_AT_0.25	0	0	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
NN_RFN	0	0.2	0	0	0	0	0	0	0	0	0
RF_AT_0.25	1	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4
SECSVM_AT_0.25	0	0.3	0.3	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
SVM	0	0	0	0	0	0	0	0	0	0	0

Table B.4: DREBIN, StratDef-Variety-Ranked, Random attacker

	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
DT_AT_0.1	0	0	0	0	0	0	0	0	0	0	0
DT_AT_0.25	0	0	0	0	0	0	0	0	0	0	0
NN	1	0	0	0	0	0	0	0	0	0	0
NN_AT_0.01	0	0	0	0	0	0	0	0	0	0	0
NN_AT_0.1	0	0	0	0	0	0	0	0	0	0	0
NN_AT_0.25	0	0	0	0	0	0	0	0	0	0	0.333333
RF	0	0	0	0	0	0	0	0	0	0	0
RF_AT_0.1	0	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.333333
RF_AT_0.25	0	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.333333
RF_RFN	0	0	0	0	0	0	0	0	0	0	0
SECSVM_AT_0.1	0	0	0	0	0	0	0	0	0	0	0
SECSVM_AT_0.25	0	0	0	0	0	0	0	0	0	0	0

Table B.5: SLEIPNIR, StratDef-Best-GT, Medium attacker

B.2 Extended Results

The extended results are located in the following repository: https://osf.io/93yzt/?view_only=bac46b0ab58b42758a133ac48f36b017

APPENDIX C

EFFECTIVENESS OF MOVING TARGET DEFENSES AGAINST ADVERSARIAL ATTACKS

C.1 Effectiveness of Attacks Against Models with Fewer Features

We evaluate the performance of our gray-box query attack strategy against two vanilla neural network models to demonstrate the effectiveness of attacks when the feature-space is *drastically* reduced. That is, for each dataset, we train two vanilla neural network models: one with the full set of features; and another with only 500 features. For the neural network with reduced features, the features are selected by using the `SelectKBest` function of the scikit-learn library (with the `chi2` scoring function that computes the chi-squared stats between each non-negative feature and class). The table below shows that the attack performs just as well when the feature-space is significantly decreased for DREBIN and SLEIPNIR datasets.

	500 features	Full features
DREBIN	100%	98.6%
SLEIPNIR	100%	99.5%

Table C.1: Evasion rate achieved by gray-box query attack against vanilla neural networks with different numbers of features.

C.2 Extended Results

The extended results are located in the following repository: https://osf.io/nym5a/?view_only=4ba9b399086c4f7cad65a6a4e8da83e

C.3 Varying Training Data Size in Gray-box Transferability Attack

We evaluate how the attack performs when the substitute models are trained with different sizes of training data (% of the original size). For DREBIN, the average evasion rate decreases between 25% to 75% sizes, after which it increases once 100% of the training data is used to construct substitute models. Meanwhile, for SLEIPNIR, we observe peak attack performance at 25% training data, with this decreasing as the training set size increases.

The evasion rate against DeepMTD and MTDeep remains consistent, with only the performance against other models decreasing drastically. This supports the idea that overfitting may be occurring, such that the substitute models are unable to generalize effectively, meaning that the generated adversarial examples are less transferable. The disparity between the trends in DREBIN and SLEIPNIR can be attributed to the differences and nuances of each dataset relating to the different-sized attack surfaces of each. The black-box transferability attack still performs better. We primarily attribute this phenomenon to the suggestions made earlier: the substitute models capture the oracle’s traits and behavior better, which means that adversarial examples for the black-box substitute models transfer better to the oracle.

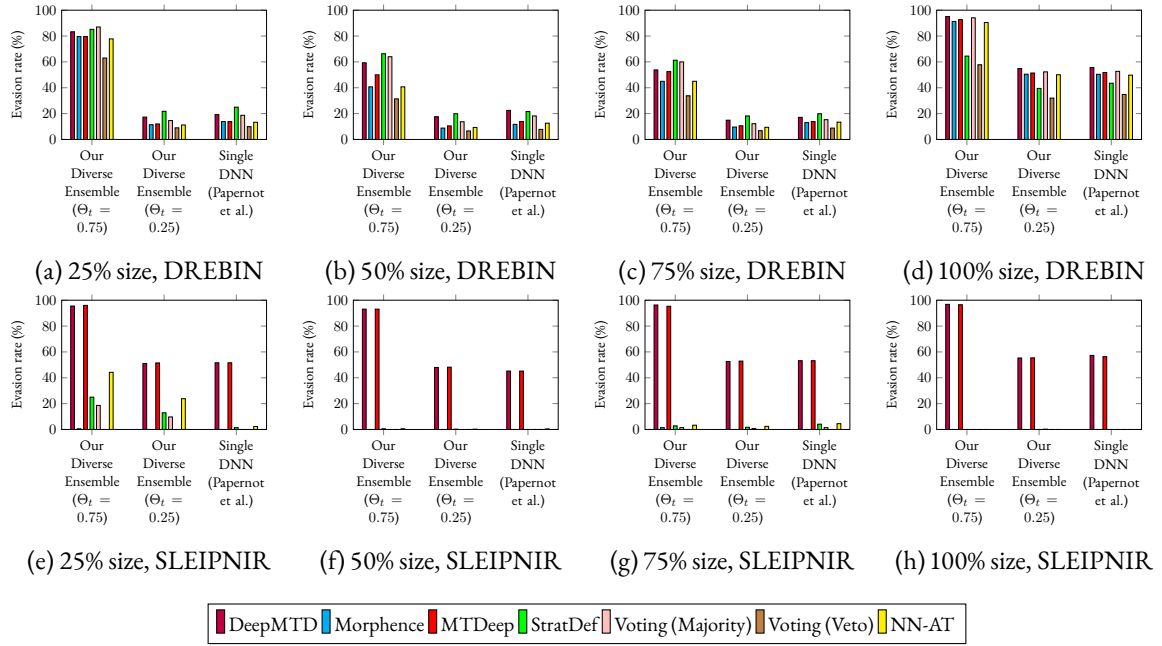


Figure C.1: Evasion rate vs. sizes of training data.

APPENDIX D

STATEFUL DEFENSE AGAINST ADVERSARIAL QUERY ATTACKS

D.1 Query Attack Strategies

We modify attack strategies from prior work on adversarial attacks in malware detection [182]. These are based on a software transplantation approach where an attacker makes perturbations based on benign samples. We modify prior attack strategies by transplanting multiple features per iteration (rather than one perturbation per iteration). The differences between the black-box and gray-box strategies lie in how perturbations are applied. The black-box attack selects the features to perturb in a randomized manner, while the gray-box attack perturbs features based on their frequency in benign samples in a heuristically-driven approach.

Algorithm 1 Black-box query attack: for oracle O , malware sample X , set of (randomly-ordered) benign features F , maximum permitted queries n_{max} .

Input: O, X, F, n_{max}

```
1:  $X' \leftarrow X, n \leftarrow 0$ 
2: while  $O(X') = 1 \ \& \ n < n_{max} \ \& \ n < F.length$  do
3:    $X' \leftarrow AddFeature(X', F[n])$ 
4:    $r \leftarrow RandomInteger(0, Length(F))$ 
5:    $F_r \leftarrow ChooseRandomFeatures(r, F)$ 
6:    $X' \leftarrow AddFeatures(X', F_r)$ 
7:    $X' \leftarrow ValidatePerturbations(X, X')$ 
8:    $n \leftarrow n + 1$ 
9:   if  $O(X') = 0$  then
10:     return Success
11: return Failure
```

Algorithm 2 Gray-box query attack: for oracle O , malware sample X , vector of sorted benign features $\vec{s}$, maximum permitted queries n_{max} .

Input: $O, X, \vec{s}, n_{max}$

```

1:  $X' \leftarrow X, n \leftarrow 0$ 
2: while  $O(X') = 1 \ \& \ n < n_{max} \ \& \ n < \vec{s}.length$  do
3:    $X' \leftarrow AddFeature(X', \vec{s}[n])$ 
4:    $r \leftarrow RandomInteger(0, Length(\vec{s}))$ 
5:    $F_r \leftarrow ChooseRandomFeatures(r, \vec{s})$ 
6:    $X' \leftarrow AddFeatures(X', F_r)$ 
7:    $X' \leftarrow ValidatePerturbations(X, X')$ 
8:    $n \leftarrow n + 1$ 
9:   if  $O(X') = 0$  then
10:     return Success
11: return Failure

```

For the adaptive attack against MalProtect, the number of features that are added in a single perturbation is capped, and from the *removable features* of the query, $p\%$ of features are removed. This is to make queries as distinct from each other as possible while remaining within the general distribution of other queries.

Algorithm 3 Adaptive attack: for oracle O , malware sample X , vector of sorted benign features $\vec{s}$, maximum permitted queries n_{max} , p percentage of features to remove, m maximum features to add in each iteration.

Input: $O, X, \vec{s}, n_{max}, m$

```

1:  $X' \leftarrow X, n \leftarrow 0$ 
2: while  $O(X') = 1 \ \& \ n < n_{max} \ \& \ n < \vec{s}.length$  do
3:    $X' \leftarrow AddFeature(X', \vec{s}[n])$ 
4:    $r \leftarrow RandomInteger(0, m)$ 
5:    $F_r \leftarrow ChooseRandomFeatures(r, \vec{s})$ 
6:    $X' \leftarrow AddFeatures(X', F_r)$ 
7:    $X' \leftarrow RemoveFeatures(X', p)$ 
8:    $X' \leftarrow ValidatePerturbations(X, X')$ 
9:    $n \leftarrow n + 1$ 
10:  if  $O(X') = 0$  then
11:    return Success
12: return Failure

```
