On Transferability of Adversarial Examples on Machine-Learning-Based Malware Classifiers

Yang Hu

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Science

Wenjing Lou, Chair

Yimin Chen

Chang-Tien Lu

May 4, 2022

Falls Church, Virginia

Keywords: malware detection, adversarial example attack, transferability, machine learning

Copyright 2022, Yang Hu

On Transferability of Adversarial Examples on Machine-Learning-Based Malware Classifiers

Yang Hu

(ABSTRACT)

The use of Machine Learning for malware detection is essential to counter the massive growth in malware types compared with the traditional signature-based detection system. However, machine learning models could also be extremely vulnerable and sensible to transferable adversarial example (AE) attacks. The transfer AE attack does not require extra information from the victim model such as gradient information. Researchers explore mainly 2 lines of transfer-based adversarial example attacks: ensemble models and ensemble samples.

Although comprehensive innovations and progress have been achieved in transfer AE attacks, few works have investigated how these techniques perform in malware data. Besides, generating adversarial examples on an android APK file is not as easy and convenient as it is on image data since the generated AE of malware should also remain its functionality and executability after perturbation. Therefore, it is urgent to validate whether previous methodologies could still have their effect on malware considering the differences compared to image data.

In this thesis, we first have a thorough literature review for the AE attacks on malware data and general transfer AE attacks. Then we design our algorithm for the transfer AE attack. We formulate the optimization problem based on the intuition that the contribution evenness of features towards the final prediction result is highly correlated to the AE transferability. We then solve the optimization problem by gradient descent and evaluate it through extensive experiments. Implementing and experimenting with the state-of-the-art AE algorithms and transferability enhancement techniques, we analyze and summarize the weaknesses and strengths of each method.

On Transferability of Adversarial Examples on Machine-Learning-Based Malware Classifiers

Yang Hu

(GENERAL AUDIENCE ABSTRACT)

Machine learning models have been widely applied to malware detection systems in recent years due to the massive growth in malware types. However, these models are vulnerable to adversarial attacks. Malicious attackers can add some small imperceptible perturbations to the original testing samples and mislead the classification results at a very low cost. Research on adversarial attacks would help us gain a better understanding of the attacker's side and inspire defenses against them. Among all adversarial attacks, the transfer-based adversarial example attack is one of the most devastating attacks since it does not require extra information from the targeted victim model such as gradient information or query from the model. Although plenty of researchers has explored the transfer AE attack lately, few works focus on malware (e.g., Android) data. Compared with image data, perturbing malware is more complicated and challenging since the generated adversarial examples of malware need to remain functionality and executability. To validate how transfer AE attack methods perform on malware, we implement the state-of-the-art (SOTA) works in this thesis and experiment with them on real Android data. Besides, we develop a new transfer-based AE attack method based on the contribution of each feature for generating AE. We then do comprehensive evaluations and draw comparisons between SOTA works and our proposed method.

Acknowledgments

I would like to thank a lot of people for helping me with this research project, without whom I would not have been able to complete this thesis. Firstly, I would like to express my sincere gratitude to my mentor Dr. Wenjing Lou for her continued guidance, feedback, and encouragement throughout my whole time at Virginia Tech. I am deeply honored to work with such a professional and brilliant researcher. I would like to sincerely thank Dr. Yimin Chen, who is also my committee member for the thesis. Without his help, expertise, and encouragement, it has not been possible for me to achieve my educational goals. I would also like to thank Dr. Chang-Tien Lu for serving on my thesis committee, and for his brilliant suggestions and comments. I would also thank to all my friends and colleagues in the CNSR lab. Their insights and advice were influential and essential throughout the thesis writing process.

This thesis and project would not have been possible without the support of the US National Science Foundation under grant CNS-1837519, the Office of Naval Research under grant N00014-19-1-2621, and the Virginia Commonwealth Cyber Initiative (CCI).

Personally, I want to thank my parents for always believing in me and supporting my dreams. Through their love, support, and encouragement, I've grown and developed.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Recent years have seen groundbreaking advancements in machine learning (ML) in both academia and industry. These achievements have driven the wide applicability of machine learning across many application scenarios as well. Particularly, machine learning has been widely used in security domains such as network intrusion detection and malware detection. Compared with the traditional signature-based detection methods, machine-learning-based detection systems not only exhibit high detection accuracy but also show the capability of detecting unseen and/or zero-day attacks due to their big-data-driven nature.Meanwhile, studies show that these systems achieve high detection accuracy even when the attackers use code concealing techniques to evade the detection systems [8, 13, 16].

Although there is no doubt that learning-based detection systems outperform their predecessors and have become the mainstream for many security applications, they suffer from the same vulnerabilities that most machine learning models do.Adversarial examples attack

(AE attack) is one of the most devastating attacks on machine learning models because it is difficult to defend against. It was first proposed in [24]. By adding small (negligible) perturbations to a testing sample, the trained model will misclassify it to the attacker's targeted class. For example, the attacker would be able to modify a piece of malware slightly so that it would be classified as "benign software" with a high probability. Considering that we heavily rely on security applications to protect our data and infrastructure, adversarial example attack thus poses an urgent threat to us.

What's worse, researchers have shown that an attacker is able to generate adversarial examples without knowing the victim model by using a 'substitute mode' and achieves a high attack success rate as well. This is the so-called "black-box transfer attack" [39, 41]. In effect, such an attack shows that the adversarial examples generated from one machine learning model, e.g., the substitute model, are also effective for another model, e.g., the victim model, provided they are somehow related. One example is that the substitute model and the victim model are obtained from the same training set with different training algorithms or settings. We refer to such a feature of adversarial examples as 'transferability'. A higher transferability corresponds to that a high portion of generated adversarial models from the substitute model are effective for the victim model. The feasibility of the black-box transfer attack greatly lowers the assumptions on the attacker, making the adversarial example attack a practical one on real-world machine learning models.

Therefore, it is necessary and important for us to better understand the adversarial example attack on security applications so as to keep improving the robustness of these applications. One research gap is that while adversarial example attack has been studied in the computer vision domain extensively, it is surprisingly quite under-researched in security/cybersecurity sectors. Without a doubt, it is even more urgent to investigate the performance of adversarial example attacks on security applications due to their peculiarity and criticality. Therefore,

in this thesis, we strive to look into the adversarial example attack on security applications by focusing on investigating the transferability of adversarial examples on malware detection systems. We believe the research findings here would help advance our understanding of the attack itself and defenses against it.

1.2 Research Challenges

As mentioned above, the goal of the transfer AE attack is to increase the probability that the adversarial examples generated from a substitute model are effective for the victim model as well. This leads to the main research challenge for transfer AE attack, i.e., the victim model could be totally unknown to the attacker. Such an adversary model is referred to as the "black-box model". Note that typically in order to generate an effective adversarial example for a specific victim model, the attacker needs to have access to the victim model (i.e., a typical AE attack adopts the "white-box" adversary model). To solve such a challenge, the generated adversarial examples in the transfer AE attack need to be effective for a range of similar victim models rather than a specific one.

Generating adversarial attack perturbation is not as natural as it is in the malware domain. In the computer vision domain, the attacker could add perturbation in continuous space and only need to guarantee the pixel values would still be in $[0, 1]$ interval after adding them up. However, the attacker should worry about the executability and the functionality after adding the perturbation to the original malware sample. The generated AEs should successfully evade the malware classifier and still have their malicious functionality.

Machine learning-based malware detection systems are varied, from SVM, and LR, to DNN, which would very likely result in different decision boundaries. Also, machine learning models always have the tendency on overfitting in the iterative training process. Some features would

be overemphasized in over-training while others would be ignored due to under-training. They all result in AEs generated from the substitute model being very likely to "overfit" to this one and fail on the victim target one, i.e., low transferability[15, 49, 51].

1.3 Contributions

We summarize our contributions in this thesis as follows.

1. We put forward a new method applying Evenness Score for improving AE transferability on the malware domain. We formulate the optimization problem and utilize the gradient descent to solve it.

2. We fill the gap in how previous works perform as transfer attacks. We investigate the effectiveness of ensemble model(EM) and ensemble sample(ES) techniques from CV and transplant them to the malware domain.

3. We implement and evaluate our proposed method on the real Android APK dataset Drebin. In the thesis, we do comprehensive experiments to validate our method. We discuss in which scenarios our techniques work well or not and compare the transferability results with other state-of-the-art AE transferability enhancement algorithms(ES and EM).

1.4 Organization of the Thesis

Chapter 2 introduces existing related research on the adversarial example attack on malware classifier, including adversarial example attacks on static victim models and on dynamic

victim models. Then, we also investigate the most related works about the transfer AE attack on the malware domain in this chapter.

Chapter 3 provides the background details about machine learning models, malware classifiers, the dataset we used in the thesis, and popular adversarial example attacks.

In Chapter 4, we discuss the adversarial model and the system model. The adversarial model consists of the attacker's goal, attacker's capability, and the attacker's strategies. The system model presents the general transfer AE attack process and the details about each component.

In Chapter 5, we develop our methodology for transfer AE attack. We introduce our intuition and build the evenness-score-based optimization algorithm. We present the details of our algorithm in this chapter.

In Chapter 6, we provide a thorough evaluation of our algorithm on a real Android dataset. Considering different substitute models and the victim models, we verify the effectiveness and the robustness of our algorithm compared with the state-of-the-art techniques.

Lastly, in Chapter 7, we conclude this thesis by summarizing the experimental results and discussing the future works.

Chapter 2

Related Work

In this chapter, we focus on the two lines of research that are closely related to this thesis: adversarial example (AE) attacks on machine-learning-based malware classifier/detection systems and AE transferability in computer vision and malware. To begin with, we survey current AE attacks on the two primary types of malware classifiers: static classifier and dynamic classifier. Next, we will go through the research on AE transferability in computer vision and malware as well to understand the current status.

2.1 Adversarial Example Attacks on Malware Classifiers

Depending on the feature representation of the victim malware detection system, adversarial example attacks in the malware domain can be divided into two categories: attacks on static victim models and attacks on dynamic victim models.

2.1.1 AE attacks on static victim models

Static malware classifiers usually analyze and examine the application and related objects without executing the software. Statics malware classifiers exclude the potential threats overhead and only need a small run-time.

For Android data, the static analysis program will first try to resolute the APK file, which is the Android application installation package. A typical APK file will include AndroidManifest.xml, smali files, etc... Useful features could be obtained from those files for an software to be examined. Static based malware classifiers often analyze the permission of applications[2, 4, 28, 44, 45, 57], analyze API calls [1, 20, 26, 36, 54] or network addresses [8, 34, 65] and construct machine learning detectors based on these static features before they are actually executed. Static analysis is an important and preliminary step for malware detection. It is crucial for keeping malware from spreading on large-scale systems.

Since the adversarial example attacks have been well-explored in the computer vision domain in recent years, some researchers have also studied how this new attack perform on machine learning-based malware detection system. Most of the existing AE attacks on the malware detection system are designed for the static-based malware classifier. In [25], Grosse et al. were the first to launch adversarial examples attack on the static malware detectors.

It attacks the different DNN Android malware classifiers using the gradient perturbation method (FGSM)[23]. In their work, the perturbation could only be added in the Manifest file so as to not interfere with other features in the features set as many as possible. In this way, the modified malware still retains its semantics. The attack success rate is about 50% to 84% according to DNN models with different model architectures and parameter settings. Many related researchers follow this work in the way of additive techniques but use different gradient information. Biggio et al.[9] try to evade the malicious PDF detection system using a gradient-based approach by adding new features. Based on the analysis of API calls, Rosenberg et al. [43] proposed a black-box attack against machine learning-based malware detectors on Windows operating system. The attack algorithm added non-operational system calls (extracted from harmless software) iteratively to the binary code. In [27], Hu et al. implemented the grey-box AE attack on the RNN-based malware detectors by additive API calls. Recently, Anderson et al. [6] used reinforcement learning to evade the malware detection system by choosing from a predefined list of transformations that preserve semantics. Several gradient-based AE attacks[31, 32] are designed typically for the MalCov arthitecures[5] for PE files. Except for the gradient information, in [64], Xu et al. utilized the heuristics algorithms to find the optimal manipulation of PDFs while maintaining necessary syntax.

However, none of the above works have explored the AE transferability property on malware data. Compared with their research, our attack is mainly focused on the transferability property of AE on Android malware. We also examine the factors of different substitute models and the victim targeted models contributing towards the attack performance. Besides, we adapt the popular transfer AE attacks from the computer vision domain to the malware domain and compare them with our work from comprehensive aspects.

2.1.2 AE attacks on dynamic victim models

The dynamics analysis technique for malware detection is more complicated and time-consuming compared with the static method. By running the software in a live environment, often a secure sandbox or test environment, the detectors can keep a record of executed and loaded code as well as any changes made to internal files, directories, and settings. Typical dynamic analysis for android malware detection often need to examine system calls [10, 11, 55, 61], API calls [60, 63], network traffic [7, 21, 22], and CPU data [3, 46, 47]. To our best knowledge, few works have explored the AE attacks on dynamics classifiers since perturbing dynamics gathering features is much more challenging than static features[50]. In [49], Song et al. proposed a grey-box-based framework to generate AEs for PE malware classifiers and Anti-Virus engines regardless of statics or dynamics detectors based on reinforcement learning. They pre-define Macro/Micro features sets and iteratively change the modification from the Macro feature set to the Micro set to gradually minimize the perturbation range. In this thesis, we only consider the static classifiers and may leave the AE attack on dynamics detectors for future works.

Most of the above research is based on the white-box adversary model and therefore cannot be applied to the scenarios we are looking into. Besides, the methods in [27, 49] proposed the grey-box AE attack framework without requiring white-box access. However, their methods need to query the targeted victim models frequently to adjust the perturbation in rounds. By contrast, our attack does not require for that and we focus on trade-offs between the transferability of generated AEs and the attackers' capability.

2.2 Transfer Adversarial Example Attacks on Malware Classifiers

2.2.1 Transfer AE attacks

It has been proved that machine learning models could be extremely vulnerable to adversarial example attacks. Researchers in the community have been looking into how to synthesize powerful adversarial samples so that they could be served as the spur for the development of effective defenses. There are generally two lines of AE attacks studies: the white-box attack and the black/grey box attack, which assume the attackers have perfect knowledge or are unspecific about the victim models, respectively. Under the black/grey-box assumptions, these researches are more practical and meaningful in the real world. Besides, in black or grey-box settings, transfer-based AE attacks have gained increasing interests recently due to their high practical applicability, where attackers craft adversarial samples based on local source models and directly harness the resultant adversarial examples to fool the remote black-box victims [59].

The transferability of AEs was first discovered by Goodfellow et al. in [23]. They had the observation that the adversarial examples would have higher transferability in the case that the intersecting or overlapping boundaries are often learned by 2 models. Later, in [39, 41] researchers found that even if the very different model structures among SVMs, logistic regression models, or neural networks, adversarial examples would successfully transferable evade these different machine learning models. Current prevailing methods to improve AE transferability can be split into 2 genres. One is inspired by the data augmentation techniques. Works in [19, 59, 62] develop direction to enhance the generated AE transferability by ensembling multiple samples together. Instead of only using the original images to gener-

ate adversarial examples, they apply data augmentation to the input images in iterations as the input to generate the AE at the base FGSM[23] algorithms. Another genre increases the AE transferability by utilizing the comprehensive gradients information from ensembles of models technique[18, 35, 56]. In their works, they concluded that the ensembled model would provide a lower-variance model since it achieved a smoother and stabler decision boundary.

However, although many researchers have recognized the importance of the transferability property of AE, none of the works have examined how these techniques perform in the malware domain. Compared with computer vision data, there are more limitations and restrictions when adding perturbation on malware data since it needs to remain functionality and execution. In this thesis, we implement the 2 works[19, 35] in the two lines to verify the effectiveness of the ensemble model and the ensemble sample technique respectively on malware data and compare them with our attack.

2.2.2 Transfer AE attack on malware classifiers

It has been well-recognized that the property of AEs' transferability should be taken more seriously. However, to the best of our knowledge, there is little research on how the above transfer attacks perform in malware classifiers not to mention proposing novel transfer AE attacks in this domain. To our best knowledge, [15, 49, 51] explore the success rate of the transfer AE attack a little bit and only show poor performance compared with that in the CV domain. Besides, their works never investigate the previous transferability enhancement techniques. In this thesis, we will demonstrate the limitations of previous transfer attacks on malware classifiers and more importantly propose our novel transfer attack which utilizes evenly-distributed adversarial perturbations to improve the transferability on unknown victim model(s). We compare our algorithm with SOTA transfer AE attacks in different

situations and experimental settings. We evaluate and conclude the results in chapter 6 and chapter 7. We hope our work on the transfer AE attack in the malware domain could shed the light on the better defense and security in the malware detection research community.

Chapter 3

Background of Transfer Adversarial Example Attacks on Malware Classifiers

In this chapter, we mainly focus on introducing the background of transfer AE attacks on malware classifiers. In particular, we will touch on ML-based malware classifiers, AE attacks on ML models, and transfer AE attacks.

3.1 Machine learning models and Notation

Here we fix the notations we use throughout this thesis and list them in the following table. As the table 3.1 shows, we use the D_{train} to indicate the training set used by the targeted victim model and the attacker's substitute model. We assume that the victim and the attacker use the same training set in our experimental setting in the following chapters. D_v and D_{test} represent the validation and testing set respectively. We use $f(\cdot)$ as a

Table 3.1: Notations adopted in this thesis.

D_{train}, D_v, D_{test}	Training, validation, and testing dataset.
$f(\cdot)$	A general machine learning model
$f_s(\cdot), f_t(\cdot)$	Substitute model and the targeted victim model.
x, x'	The feature vector of a malware and the corresponding AE.
y	Ground truth label of x or x'.

general machine learning model, with the subscript $f_s(\cdot)$ and $f_t(\cdot)$ indicating the substitute model used by the attacker and the victim model the attacker target. Since the attacker will generate the perturbation on the original malware sample and try to evade the victim model, we use the notation x representing the original malware sample and x' indicating the corresponding adversarial example after adding the perturbation. The ground truth label of sample x is y.

3.2 Machine-learning-based Malware Classifiers

3.2.1 Dataset and Data Processing

Drebin. Drebin is one of the most popular datasets used for evaluating malware classifiers [8]. The dataset includes SHA256 values of 129,013 android applications, of which 123,453 are benign and 5,560 are malicious. Based on the given SHA256 value, we collect the APK files from the APK markets, including VirusTotal for the malware APKs and GooglePlay store or AppChina for benign APKs.

Data Processing Since machine learning models only take numerical values as input, we need to extract numerical features from the original APK file. Here we also follow the work in [8], which is an effective static analysis method. An APK file must contain 2 files: AndroidManifest.xml and classes.dex. Additional XML files and resource files are defined

Figure 3.1: APK file

by the owner of this APK and used for application layout and multi-media contents. Most Android malware classifier researches[25, 33] follows work in [8] only analyze classes.dex file and AndroidManifest.xml file. We follow them and briefly introduce the 2 files below.

- *AndroidManifest.xml* is the most basic entry file of the APK file. It carries all the essential and overall information about Android applications, such as the name of the Java package, a description of components, the list of hardware components and permissions requested by the application to work (e.g., Internet access).

- *Class.dex* compiled all the source code files of the Android application. The Classes.dex may include some suspicious API calls which would visit or request sensitive data such as contact information or personal resources. It also includes restricted API calls related to the system for which the function requires permission (such as if it can use the Internet service). Furthermore, this file may contain references to some network

addresses to which applications can connect.

Table 3.2: Overview of the feature sets.

	manifest		dexcode
S_1	Hardware components	S_5	Restricted API calls
S_2	Requested permissions	S_6	Used permission
S_3	Application components	S_7	Suspicious API calls
S_4	Filtered intents	S_8	Network addresses

Feature Extraction We follow work in Drebin[8] framework to statically analyze Android applications. Drebin constructs a features space based on the 2 files above and it extracts and splits features into 8 categories, as listed in table 3.2. Subsets S_1 to S_4 are extracted from AndroidManifest.xml and S_5 to S_8 are extracted from Class.dex file. Specifically, the S_1 set consists of the APK requests for hardware components such as the camera, touchscreen, or the GPS module of the smartphone. S_2 set includes all the permissions the APK asks to access security-relevant resources such as requesting and sending premium SMS messages. S_3 set contains the features that are related to the application components (e.g., activities, receivers, service, etc.). S_4 collects all the features related to Inter-process and intra-process communication on Android. S_5 contains features related to sensitive system API calls that will not work without permissions from users or root privileges. S_6 set involves all the features that correspond to the used permissions. S_7 set includes all the features that can be related to API calls leading to sensitive data or resources on a smartphone. S_8 set contains network addresses as features including IP addresses, hostnames, and URLs found in the disassembled code.

An APK file $z \in Z$ then will be mapped in to a feature vector $x \in X$ through a mapping function $Z \rightarrow X$, where $x = (x^1, ..., x^d)^T \in X = \{0, 1\}^d$. Each dimension of x indicates whether this feature exists in the APK file z. A feature vector extracted and encoded from an APK would present in this way:

$$\boldsymbol{x} = \Phi(\boldsymbol{z}) \mapsto \begin{pmatrix} \cdots \\ 0 \\ 1 \\ \cdots \\ 1 \\ 0 \\ \cdots \end{pmatrix} \begin{matrix} \cdots \\ \texttt{permission::SEND\_SMS} \\ \texttt{permission::READ\_SMS} \\ \cdots \\ \texttt{api\_call::getDeviceId} \\ \texttt{api\_call::getSubscriberId} \\ \cdots \end{matrix} \begin{matrix} \\ \\ \left.\begin{matrix} \\ \\ \end{matrix}\right\} S_2 \\ \\ \left.\begin{matrix} \\ \\ \end{matrix}\right\} S_5 \\ \\ \end{matrix}$$

3.2.2 Constraints in Generating Malware Adversarial Examples

Now we can train machine learning models based on our APKs gathered from the SHA256 value of Drebin and the above preparation. As an attacker, we need to generate a new feature vector $\delta + x$ based on the original malware input vector x to evade the victim model. Given a malware, the attacker also wants to maintain the integrity and the malicious functionality of the original application after modification. We only consider the simple feature addition strategy:

- '0' to '1': The attacker can flip the value in the input vector from '0' to '1', meaning injecting features in the original APK file, such as adding more permission requests in AndroidManifest.xml or another system call in Class.dex file.

The '0' to '1' change is a much safer manipulation compared with the '1' to '0' change. Normally, adding more features to the original sample at AndroidManifest.xml would not affect its functionality(eg, more permission requests). Also, when the '0' to '1' change comes to the Class.dex file, it is also safe to add some dead code that has never been called or executed.

3.2.3 Static Malware Classifiers

A typical static malware classifier is depicted in Figure 3.2. The static analysis would classify the file based on various features directly extracted from the executable. Disassembly is one of the methods, which is used for extracting various features from the executables. Typical features used for malware static analysis including API calls, suspicious network addresses, OPCODES, byte sequence, or PE header[31]. Then the encoded feature vectors would be sent to the machine learning model for training or testing.

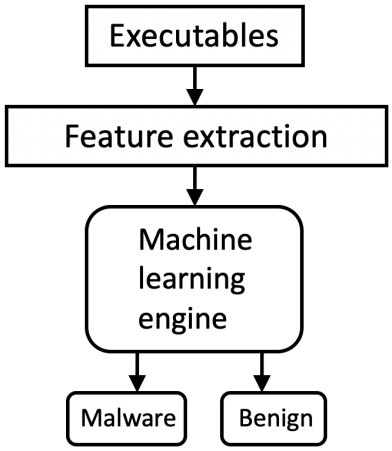


Figure 3.2: The static malware detection scheme.

3.2.4 Dynamics Malware Classifiers

A typical dynamics malware analysis needs to execute the malware in an isolated environment for analysis, as depicted in Figure 3.3. This could be a debugger or sandbox environment. The typical procedure used for Dynamics analysis includes:

- analyzing by utilizing sandbox
- Monitoring the system calls

- Monitoring the file changes
- Monitoring network activities
- Processing the monitoring

Then we will obtain specific features from the original dataset and then classify them using ML algorithms.

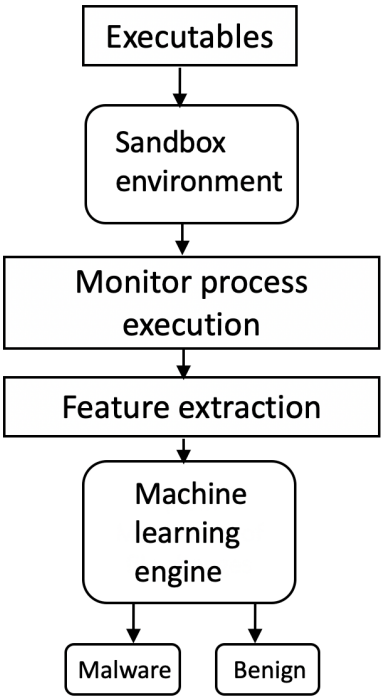


Figure 3.3: The dynamic malware detection scheme.

3.3 Adversarial Example Attacks

AE attacks refer to the family of adversarial attacks on machine learning models where they are being attacked in the testing phase. As an example shown in Figure 3.4, by adding some perturbations that cannot be detected by the human eyes (such perturbations will not affect

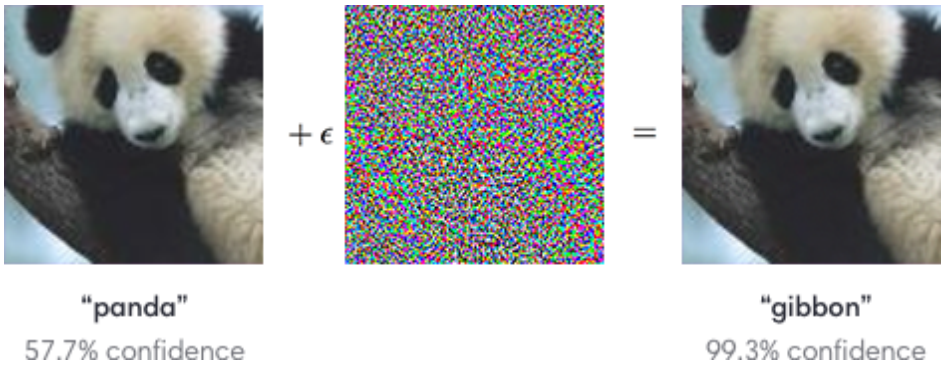


Figure 3.4: A sample of 'Panda' is perturbed and predicted as 'Gibbon' under AE attacks [24].

human recognition, the attacker can be easy to fool the model) and cause the machine to make the wrong decision.

3.3.1 Popular AE Attacks

For simplicity, we only introduce the four most popular AE attacks in the literature.

1. L-BFGS [53]. Szegedy et al. demonstrated for the first time that a neural network can be misled into misclassification by adding small perturbations to images that are imperceptible to humans. They first tried to solve the equations that would allow the neural network to misclassify at the smallest perturbation. But because the complexity of the problem was too high, they turned to solve the simplified problem, that is, finding the smallest addition to the loss function, which turned the problem into a convex optimization process.

$$\begin{aligned} \underset{\delta}{\text{minimize}} \quad & c||\delta|| + J_\theta(x+\delta, y^*), \\ \text{subject to} \quad & x+\delta \in [0,1], \end{aligned}$$

where J_θ is the loss function of the victim model and y^* is the targeted label. To find

a suitable constant c, the L-BFGS algorithm finds an approximation of c by linearly searching all cases where $c > 0$. Experiments show that the generated AEs can also generalize to and transfer from different machine learning models or different training sets.

2. FGSM [24]. The linear search method used in the L-BFGS method is expensive and impractical. Goodfellow et al. proposed a fast method called FGSM. They performed only one gradient update in the direction of the gradient sign at each pixel. Their perturbation form is:

$$x' = x + \epsilon sign(\nabla_x J_\theta(x, y)),$$

where ϵ is the magnitude of the perturbation.

3. JSMA [40]. Papernot et al. calculated the Jacobian matrix of the original sample x as follows:

$$J_f(x) = \frac{\partial f(x)}{\partial x} = [\frac{\partial f_j(x)}{x_i}]_{i*j},$$

f represents the second layer to the last layer of neural network (logit used in the last layer at the beginning, and then modified to softmax). In this way, they found that the input features of the sample x have the most significant impact on the output. Then, a small designed perturbation can cause a large change in the output, so that a small feature change can mislead the neural network. Furthermore, this work define 2 adversarial saliency maps to pick out the features pixels that are created during each iteration.

However, this method is time-consuming since computing the Jacobian matrix is slow.

4. C&W [12]. Carlini and Wagner proposed an optimization-based adversarial example attack C&W that can generate adversarial examples under the L_0, L_2, and L_{inf} norm

constraints. Similar to L-BFGS, the optimization objective function is expressed as:

$$\begin{aligned} \underset{\delta}{\text{minimize}} \quad & D(x, x+\delta) + c * f(x+\delta), \\ \text{subject to} \quad & x+\delta \in [0,1]. \end{aligned}$$

D represents the different distance norm L_0, L_2, and L_{inf}. $f(x+\delta)$ is a self-defined adversarial loss for easy solution. C&W attack can successfully attack the defensive distillation networks and also is effective against most existing defense methods.

3.3.2 Existing Transfer AE Attacks

As mentioned above, many works have explored the transferability of the AE attack. In the transfer AE attack, the attacker can train a substitute model by himself first. He will attack his own model and generate adversarial samples from it. Then the generated AE would be used to attack the targeted victim model. Since there is no difference between attackers and ordinary users and it requires mostly zero knowledge about the targeted victim model, the transfer AE attack poses the greatest threat against model security. There are mainly 2 directions to enhance the AE transferability. We introduce them here.

Ensemble-Sample Transfer Attacks. Ensemble-sample transfer attack is basically based on the data augmentation idea. It ensembles multiple samples together at first or in the middle process of generating AE to enhance the transferability of generated AE. In work[19] Yinpeng Dong et al. put forward the Translation-Invariant Attack. In particular, the proposed method generates the adversarial examples by utilizing a set of different translated samples from the original sample rather than optimizing it from a single point:

$$\begin{aligned} \underset{\delta}{\text{argmin}} \quad & \sum_{i,j} w_{i,j} J_\theta(T_{i,j}(x+\delta), y^*), \\ \text{subject to} \quad & D(x, x+\delta) < \epsilon \\ & x + \delta \in [0, 1], \end{aligned}$$

where $T_{ij}(x)$ is a translation operation that shifts the image x along two dimensions of i and j pixels respectively. Using this method, the generated adversarial examples would be more likely to successfully evade another model since they are less sensitive to the discriminative regions of the attacker's substitute model.

There are other works that have a similar idea using ensemble samples to enhance transferability such as adding Gaussian noise in Variance-reduce attack [58], random zoom, or filling in Input-diversity attack [62]. We will examine the performance of the ensemble sample technique in the malware domain in our thesis.

Ensemble-Model Transfer Attacks Improving the transferability of adversarial samples is nothing more than hoping that the adversarial samples generated for one model will also work on a different model. Therefore, the most direct idea is to integrate multiple models to generate transferable adversarial samples.

This approach was first mentioned by Yanpei Liu et al. [35]. Given a set of models $\{f_n(x)\}$ and a set of weights $\{a_n\}$, the attackers use the the ensemble model $F(x) = \sum_{i=1}^{n} a_i f_i(x)$ which is the weighted average of the set of models. The basic idea is to generate adversarial examples from $F(x)$. The ensemble-based approach solves the following optimization problem:

$$\begin{aligned} \underset{\delta}{\text{argmin}} \quad & -log((\sum_{i=1}^{n} a_i f_i(x+\delta)) \cdot \mathbf{1}_{y^*}) + \lambda D(x, x+\delta), \\ \text{subject to} \quad & x+\delta \in [0,1], \end{aligned}$$

where a_i is the ensemble weight, $\sum_{i=1}^{n} a_i = 1$, and y^* is the target label which the attacker want to mislead. This ensemble method has quickly become a basic operation enhancing AE transferability, and following research also put forward more complicated methods based on this one. We will also implement it and evaluate how the ensemble model performs in the malware detection domain in the later chapter.

Chapter 4

Adversary and System Model

To better illustrate our research, we introduce our adversary model and the corresponding system model in this chapter. The adversary model covers the attacker's goal, knowledge, and capability in the investigated attack while the system model focuses on how the victim model works and how the attacker launches transfer AE attacks on the victim model.

4.1 Adversary Model

Attackers' Goal. Here we assume the attacker's goal is to increase the transferability of adversarial example attacks (AE attacks) on malware classifiers. First of all, AE attacks can be divided into two categories: targeted attacks and untargeted attacks. The goal of targeted attacks is to force the prediction of a specific AE to a targeted class while the goal of untargeted attacks is to force the prediction to be any other classes except the correct one. We illustrate the difference between the two categories in Table 4.1 as well. Secondly, our attacker focuses on the transferability of AE attacks. In this thesis, since the malware detection system only has 2 classes (benign and malware), the targeted attack is equivalent

Table 4.1: Targeted v.s. Untargeted Attacks. Denote victim model by $y = f(x), y = 1, \cdots, n$, clean sample by x, and AE by x'. Therefore, we have $f(x) = c$.

Targeted attacks	$f(x') = t, t \neq c$ is the targeted class.
Untargeted attacks	$f(x') \neq c, f(x) = c$

to the untargeted attack. We define transferability as the probability that AEs which are crafted from one white-box model (i.e., the attacker knows the white-box model) can be applied to other black-box models (i.e., the attacker does not know the black-box models). A higher probability corresponds to a 'higher' transferability, which indicates that the AEs are more powerful (i.e., they can be applied to unknown models).

Attackers' Capability. Here the attacker's capability refers to how much information she knows about the victim model. From the attacker's viewpoint, there are four pieces of information about the victim model that are important for AE attacks. In specific, they are the training dataset denoted by D, the feature representation of a sample denoted by X, the model architecture denoted by $\hat{f}$, and finally, the model parameters denoted by $\hat{w}$. Depending on the amount of information the attacker has, we can divide the investigated transferability attacks into three categories: white-box attacks, grey-box attacks, and black-box attacks. A white-box attacker knows everything about the victim model, i.e., $\{D, X, \hat{f}, \hat{w}\}$, a black-box attacker knows nothing about the victim model, and finally a grey-box attacker knows a subset of $\{D, X, \hat{f}, \hat{w}\}$. We illustrate the difference between the two categories in Table 4.2 as well. In this thesis, we adopt a grey-box attack in which the attacker knows X and a subset of D while she does not know $\hat{f}$ and $\hat{w}$. We believe that our grey-box assumption is realistic in that an attacker should be able to obtain some training samples and X while much more challenging to obtain $\hat{f}$ and $\hat{w}$.

Attackers' Strategy. As described above, transferability attacks under the grey-box adversary model consist of two steps. In Step One, the attacker builds a substitute model for

Table 4.2: Categorization of attack capability. Denote victim model by $y = f(x), y = 1, \cdots, n$, clean sample by x, and AE by x'. Therefore, we have $f(x) = c$.

Attack Category	Attacker's Information	Difficulty Level
White-box attacks	$D, X, \hat{f}, \hat{w}$	Low
Grey-box attacks	X and a subset of D	Medium
Black-box attacks	None	Difficult

the victim model using any resources she has under her disposal including X and a subset of D. The substitute model is expected to be 'similar' to the victim model. Note that the substitute model is a white-box model to the attacker as she builds it by herself. In Step Two, for a given clean sample X, the attacker generates the corresponding AE X' with respect to the substitute model, i.e., X' is a successful AE on the substitute model. In Step Three, the attacker applies X' to the victim model to see if X' works on the victim model or not. The higher probability that X' works on the victim model, the higher transferability our attack has.

4.2 System model

4.2.1 General transfer AE attack process

In this thesis, we aim at generating adversarial examples with higher transferability to launch powerful attacks in the grey-box scenario. We assume that the attacker has zero information except for the feature representation X and the training data D of the targeted victim model. The attacker first builds a substitute model $f_s(x)$ using the feature representation X and the training data D. Assume that the attackers have a set of testing samples that have been classified as malware by the victim model. Then the adversarial examples of these testing samples would be generated on this substitute model $f_s(x)$. In such a case, the

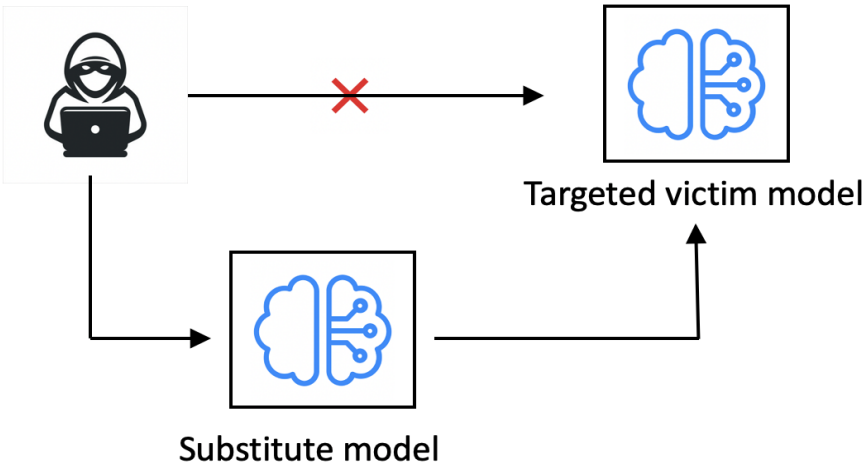


Figure 4.1: Transfer AE attacks.

victim model is a grey box for the attacker while the substitute model is a white-box one. Then the generated adversarial examples are evaluated on the victim model again to see if they can successfully evade the victim model. Note that the attacker's goal is to improve the probability that AEs generated from the substitute model could also evade the targeted victim model successfully.

Figure 4.1 illustrates the process of transfer AE attack. Rather than generating the adversarial examples directly on the targeted victim model, the attacker trains the substitute and generates AE from it for attacking the targeted victim model. The following content in this chapter will introduce the details about the targeted victim model and the substitute model discussed in this thesis.

4.2.2 Victim model

We assume our targeted victim model is a malware classifier built from APKs collected from the Androzoo repository. In this thesis, we investigate four different types of malware classifiers as follows.

1. **Support vector machine** is a linear two-class classification model. The support vector machine is trying to enlarge the interval between the two classes defined in the feature space. The basic idea is to find the optimal separating hyperplane so that it can separate points in different classes as far as possible and at the same time maintain the classification accuracy.

2. **Logistics regression** models the probabilities for binary classification problems. It uses a linear combination of features $f(x) = \omega^T x + b$ and the logistic function $g(z) = \frac{1}{1+e^{-z}}$ to map the real value from the linear function $f(x)$ to the logistic function $g(z)$ range $[0, 1]$, corresponding the probabilities to be predicted. It essentially adapts the linear regression formula to allow it to act as a classifier.

3. **Ridge regression** has the same formulation of the classification model. The only difference is that it adds the L_2 regularization on the cost function of the original logistics regression method:

$$F_{RR}(\omega) = F_{LR}(\omega) + \lambda||\omega||, \tag{4.1}$$

   where F_{LR} is the loss function of the logistics regression, ω is the parameters of the model to be optimized and λ is the hyperparameter set by the model owner. By adding this L_2 regularization on the loss function, ridge regression could reduce parameters overfitting effects caused by unbalanced contribution of parameters. Variables with minor contributions have their coefficients close to zero. However, all the variables are incorporated into the model. This is useful when all variables need to be incorporated into the model according to domain knowledge.

4. **Deep learning** is also a popular machine learning classification model applied to malware detection. It simulates how biological neurons interact with the real world. A

deep neural network consists of multilayer networks and multiple activation functions in each layer. These neurons are good at extracting hierarchy feature representations from the raw input vectors in a non-linear way. It has great capability in dealing with huge training data and complicated input features compared with traditional machine learning methods.

4.2.3 Substitute model

Under the grey-box adversary model adopted in this thesis, we build different types of substitute models in order to generate AEs against the targeted victim model. In our setting, we assume that the attacker could also use **SVM, LR, RR, and DNN** as the substitute model, which is just introduced above. Aside from that, we also examine the ensemble model as the substitute model:

- **Ensemble model** is one special kind of learning model by combining multiple learning models together as one single model, which is discussed in chapter 3. By incorporating different models and training them together, the trained ensemble model normally would have greater generalization capability and tend to have a more stable classification performance. Here for the above 4 machine learning models, we will choose 3 different models as one ensemble model as the attacker's substitute and leave out one model as the victim model each time to evaluate how the adversarial examples generated from the ensemble model perform in the following experiments.

Chapter 5

Methodology

In this chapter, we will formally introduce our transfer adversarial example attack on malware classifiers. First, we introduce the intuitions and the workflow for our proposed attack. Then we focus on the design of the attack in detail.

5.1 Research Objective

In this chapter, we introduce our attack method to generate high-transferability AEs under grey-box settings. Our method is based on the intuition that if the adversarial perturbations on a clean sample are distributed evenly across the feature vector, the corresponding AE is more likely to incur a higher transferability. Therefore, we aim to improve the transferability of AEs in our attack by optimizing the distribution of the adversarial perturbations.

5.2 Intuitions

Our intuition can be further explained as follows. Machine learning models, particularly deep learning models, tend to be overfitted as a result of iterative training [30], which makes a trained model very sensitive to a small portion of 'important' features. In other words, small perturbations over these features could lead to a significant change in the output layer, thus making a trained model vulnerable to AE attacks. In [16, 30, 37], the authors showed that machine learning models were more robust against sparse attacks if the feature importance for model prediction was distributed more evenly across the features. We use $I(x_i)$ to denote the importance of $x_i, i = 1, 2, \cdots, n$ (x_i is the i-th features in x). If $variance(I(x))$ is smaller, the corresponding ML model is likely to be more robust. On the contrary, assuming that different ML models have different sets of important features for their predictions (even under the same feature presentation X), it is intuitive that an AE with more evenly-distributed perturbations has a higher probability to be effective in an unknown model when compared to an AE with less evenly-distributed perturbations. In other words, even the distribution of perturbations across different features can improve the transferability of AE attacks. Assigning the contribution to each feature more evenly could be viewed as the resistance

against AE overfitting effect on one specific model. We will introduce our mathematical formulation below.

5.3 System Overview

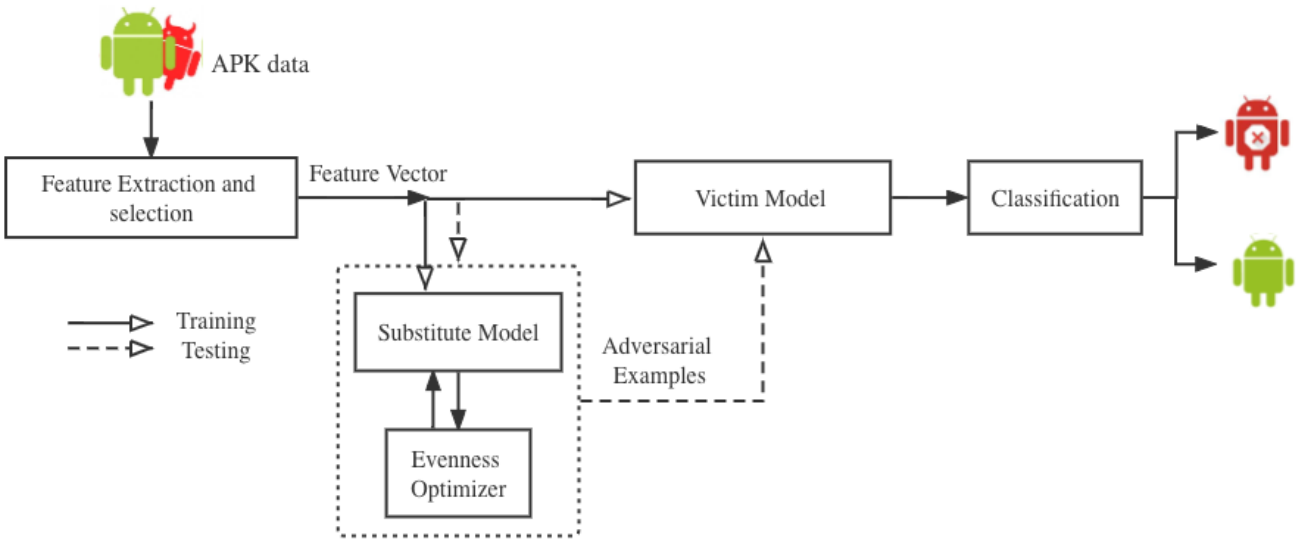


Figure 5.1: Overview of our transfer adversarial example attack.

Figure 5.1 illustrates the overview of our transfer attack, which consists of the five main steps as follows.

1. Data pre-processing. As introduced before, we process the original Android apk files into the same vector representation proposed in [8]. The resulted vector x has a size of $n_d \times 1$.

2. Substitute model training. We train a substitute model (denoted by $f_s(x)$) from a subset of D. Here D is the training set of the targeted victim model $f_t(x)$.

3. AE generation. We use AE algorithms such as L-BFGS and FGSM to generate AEs targeting $f_s(x)$. Note that $f_s(x)$ is a white-box model to the attacker.

4. Evenness optimizer. We solve a formulated objective function which is to find evenly-distributed adversarial perturbations on x. Once found, we applied them to x and obtain the final AE, i.e., x'.

5. Transferability evaluation. We feed the generated x' to $f_t(x)$, i.e., the targeted victim model, to check if x' is classified to a targeted class t by $f_t(x)$ or not, i.e., $f_t(x') = t$. If $f_t(x') = t$, our transfer attack succeeds. Vice versa.

5.4 Adversarial Perturbations with Even Distribution

5.4.1 Evenness score

The uniqueness of our transfer attack is that we aim to generate AEs with high evenness scores such that they are more likely to incur a high transferability. Our definition of evenness score can be easily illustrated in two steps. Recall that we denote a machine learning model and its input feature vector by $y = f(x)$ and x, respectively.

1. **Step One**, we use explainable machine learning techniques [37, 48] to compute the importance of each feature in x on the prediction result, i.e., $f(x)$. As a result, we obtain the contribution vector r of which r_i corresponds to the importance of x_i on $f(x)$.

2. **Step Two**, we compute an *Evenness Score*, ε, using the following equation.

$$\varepsilon = \frac{||r||_1}{n_d||r||_\infty}. \tag{5.1}$$

Here $r = \{r_i\}_{i\in[1,n_d]}, \sum_{i=1}^{n_d} r_i = 1, m$ is the number of features in x.

Based on the definition, we can draw two properties about the evenness score ε.

- ε is maximized and $\varepsilon_{max} = 1$ if and only if $r_i = \frac{1}{n_d}, i = 1, 2, \cdots, n_d$, i.e., all features have the same importance. That is, $\varepsilon_{max} = \frac{||r||_1}{n_d||r||_\infty} = \frac{1}{n_d \cdot \frac{1}{n_d}} = 1$. In this case, every feature contributes to the final prediction equally.

- ε is minimized and $\varepsilon_{min} = \frac{1}{n_d}$ if and only if $r_k = 1, r_i = 0, i \neq k$, i.e., only one of the m features has non-zero importance value. That is, $\varepsilon_{min} = \frac{||r||_1}{n_d||r||_\infty} = \frac{1}{n_d \cdot 1} = \frac{1}{n_d}$. In this case, $f(x)$ depends solely on x_k and is free from all the other features. As a result, $f(x)$ is highly sensitive to small changes in x_k.

Our intuition to boost our transfer attack is to minimize the variance of $\{I(x'_1), I(x'_2), ..., I(x'_{n_d})\}$ of the generated x's, which makes x's become more likely to succeed in attacking the targeted victim model $f_t(\cdot)$.

5.4.2 Formulating the objective function

Next, we formulate the objective function to generate AEs with optimal ε. Intuitively, AEs can be generated by solving the following objective function:

$$\begin{aligned} \underset{\delta}{\text{maximize}} \quad & \varepsilon(x+\delta), \\ \text{subject to} \quad & f_s(x+\delta) = t, \\ & x + \delta \in F^m. \end{aligned} \tag{5.2}$$

Here $\varepsilon(\cdot)$ is the even score function, x is the feature representation of an input malware, δ is the adversarial perturbations for x, $f_s(\cdot)$ is the substitute model from the attacker, t is

the targeted class, and F^m denotes all valid input feature vectors. The resulted AE x' is $x' = x + \delta$. The above objective function is to find AE x' from x so that x' is classified to a targeted class t while its evenness score $\varepsilon(x')$ is maximized. Typically, as the attacker, we aim to generate x' which is still a malware but classified by $f_s(\cdot)$ as a benign application. Since usually $f_s(x) = 1$ corresponds to a malware and vice versa, a typical t is $t = 0$. Note that the $f_t(\cdot)$ is the black-box model for the attacker, so he can only optimize the above functions on $f_s(\cdot)$.

Due to the nonlinear property of $f_s(x + \delta) = t$, Equation 5.2 is not solvable. We follow the approach in [12] and replace $f_s(x + \delta) = t$ by $g(x + \delta) \leq 0$ where $g(x)$ is defined as:

$$g(x) = max\{0.5 - f_s(x), 0\}. \tag{5.3}$$

Then the formulation of our optimization problem 5.2 is stated as follow:

$$\begin{aligned} \underset{\delta}{\text{minimize}} \quad & -\varepsilon(x + \delta), \\ \text{subject to} \quad & g(x + \delta) \leq 0, \\ & x + \delta \in F^{n_d}. \end{aligned} \tag{5.4}$$

Moreover, we further modify the objective function as follows by Lagrangian relaxation:

$$\begin{aligned} \underset{\delta}{\text{minimize}} \quad & -\varepsilon(x + \delta) + cg(x + \delta), \\ \text{subject to} \quad & x + \delta \in F^{n_d}, \end{aligned} \tag{5.5}$$

where $c \leq 0$ is a penalty constant and a hyper-parameter. Different c can lead to quite different δ thus x'. In practice, we conduct a grid search to find a better c. Intuitively, there exist some c where Equation 5.4 and 5.5 have the same solution. Since we remove

the constraint in formula 5.4, the minimum of objective in formula 5.4 could be smaller. Then we add a penal term in formula 5.5 to penalize the original objective function. There exists a c such that the optimal solution to objective 5.5 equal to the optimal solution of the objective 5.4 with the constraints.

5.4.3 Solving the objective function

Here we focus on solving Equation 5.5 covering how to find c and handle the constraints.

5.4.3.1 Finding the constant c

As mentioned above, we use a grid search to find a proper c for Equation 5.5. Specifically, we start from 0.01 to 1000. For each c, we then use stochastic gradient descent (SGD) to obtain δ_p, i.e., the optimal δ. Note that it is possible that we cannot obtain δ_p for some x when solving Equation 5.5. If c exceeds the threshold we set(1000 here) and still cannot meet the requirements, we will abort search and show no result for this sample x. We simply report such cases as failure. Also, the more precise solution could be found by reducing the interval range between different c.

5.4.3.2 Box constraint

As introduced in Chapter 3, x_i for an Android apk corresponds to whether or not a specific permission or API call is declared in the corresponding *AndroidManifest.xml* file. As a result, x_i is either 0 (i.e., not declared) or 1 (i.e., declared). So we replace the constraint in formula 5.5 as:

$$x + \delta \in \{0, 1\}^{n_d}, \tag{5.6}$$

which is called as "box constrait". Following [12], we set another form constrait in Equation 5.5:

$$\delta_i = \frac{1}{2}(\tanh(w_i) + 1) - x_i, \tag{5.7}$$

where w_i is the replacement variable for the ith dimension of optimization variable δ_i. Since $0<\frac{1}{2}(<tanh(w_i) + 1)<1$, the new modified vector $\delta_i + x_i$ would always fall into valid range. Instead of optimizing δ directly, we modify the input vector x on variable w of function $tanh$ and valid the box constraints automatically.

5.4.3.3 L_{inf} and L_1 adjustment

.

Based on optimization formulation and evenness score function defined above, now we have the optimization objective:

$$minimize \;\; cg(x + \delta) - \frac{||r||_1}{n_d \cdot ||r||_\infty}. \tag{5.8}$$

We notice that directly solving Equation 5.8 does not generate AEs well. One possible reason is that it is hard to implement the gradient search on one component in Evenness score $||r||_\infty$ directly. Since the norm infinity $||r||_\infty$ only cares about the only one dimension with the largest value, it penalizes only one dimension at one time. However, there is the possibility that the value of r_j could increase when we penalize r_i at one time. It is possible that only a few dimensions could be modified and they just change back and forth, without any optimization progress in gradient descent. Also, we normalize the $||r||_1$ each iteration to make it always equal to 1 to further simplify the optimization formulation. We want the direction of the gradient descent toward the decrease of all dimensions with a "large value".

As a remedy, we modify Equation 5.8 as follows.

$$\underset{\delta}{\text{minimize}} \quad g(x+\delta) + \frac{\sum_i [r_i - \lambda]}{n_d}. \tag{5.9}$$

Here λ is a hyper-parameter used to penalize all $r_i \geq \lambda$, thus avoiding only penalizing one dimension at one iteration. We also introduce a decay factor, $\gamma \leq 1$, to λ, i.e., $\lambda_{k+1} = \lambda_k^{\gamma}$ so that the algorithm could optimize more r_i gradually and smoothly. k denotes the k-th iteration round.

5.4.4 Summary

Algorithm 1 The Proposed Algorithm

Input: training dataset D_{train}, validation dataset D_v, testing set D_{test}, maximum threshold c_{max}, minimum threshold c_{min}, penalizer λ, decay factor γ, malware example $\hat{x}$ to be modified
Output: adversarial example $\hat{x}'$

$f_s(\cdot) \leftarrow train\_model(D_{train}, D_v, D_{test})$ # Train the substitute model
$\hat{x}' \leftarrow \hat{x}$
$c \leftarrow c_{min}$
while $c < c_{max}$ **do**
  $r \leftarrow contribution\_compute(f_s(\cdot), \hat{x}')$ #Compute the evenness score
  $\delta \leftarrow gradient\_descent(f_s(\cdot), \hat{x}', \lambda, r)$ # Optimization step based on Equation 5.9
  $\hat{x}' \leftarrow \hat{x} + \delta$
  if $f_s(\hat{x}') == 0$ **then**
    break # Successfully evade the substitute model
  end if
  $\lambda \leftarrow \lambda^{\gamma}$
end while
return $\hat{x}'$

We summarize how to generate an AE in our transfer attack and illustrate it in Algorithm 1. Given one input feature vector $\hat{x}$ and its label $\hat{y}$, our attack first generates the corresponding AE $\hat{x}'$ and evaluates it through the following steps.

1. Solve the optimization problem in equation 5.2 by Algorithm as described in details in section 5.4 .

2. First check if the generated AE successfully evades the substitute model. If successful, the attacker sends it to the targeted victim model. Otherwise, the generated AE would be viewed as the failed one and be dropped.

3. Then check the if the success AE could also evade the victim model as well. Transfer AE attack succeeds if it works.

4. Continue with the next malware AE generation.

Chapter 6

Performance Evaluation

In this chapter, we want to figure out how the proposed transfer AE attack works on benchmark datasets. To do that, we first introduce how we deal with data set including dataset splitting and feature selection. Then we introduce the experimental settings for evaluating our algorithm and validating the experimental results properly. Finally, we compare the results of our method with several baseline methods.

6.1 Dataset

Dataset splitting As mentioned in Chapter 3, we obtain 5,560 malware examples and 123,453 benign examples from Drebin repository. To train a machine learning model, we split all the malware and benign samples into three sets: a training set (60%), a validation set (20%), and a testing set (20%). The training set and validation set are purely used for model training. After all the models are trained, we further select 100 pieces of malware in the testing set randomly, which are classified as malware by the victim model and the substitute model.

Feature selection. In Drebin, each sample has roughly 545,000 different features in total($n_d = 545,000$). We further leave out features with low frequencies and select 10,000($n_d = 10,000$) features at top frequencies in our dataset as the input vector for machine learning models.

6.2 Performance Metrics

Reflecting on the adversarial model we discussed in Chapter 4, the attacker trains a substitute model and generates adversarial examples on it while aiming to evade the victim model. Therefore, we use two evaluation indicators to evaluate the attacks on the substitute model and the victim model, respectively.

6.2.1 Success rate

Assuming the attacker try to generate adversarial examples on the substitute model and test on the victim model, attack success rate is defined as follow:

$$SR = \frac{N_s}{N}, \tag{6.1}$$

where N_s is the number of generated adversarial examples that successfully evade the substitute model. Success rate evaluates the attack algorithm's strength on the substitute model.

6.2.2 Transferability

After generating adversarial examples on the substitute model, the attacker wants to examine how these generated adversarial examples perform on the victim model. It is evaluated by the transferability:

$$T_{SR} = \frac{N_v}{N_s}, \tag{6.2}$$

where N_v is the number of generated adversarial examples evade the attacker's substitute model $f_s(\cdot)$ and also successfully evade the targeted victim model $f_v(\cdot)$.

6.3 Training the victim and substitute model

According to the adversarial model introduced in Chapter 4, we need to train two models for each experiment: the victim model and the attacker's substitute model. Note that the attacker's knowledge is formed as the quadruple $\{D, X, \hat{f}, \hat{w}\}$. For the training set D, We train the victim model and the substitute model using the full training set of Drebin. In our experiments, we assume the attacker and the victim use the same feature representation X.

For each experiment, we choose one victim model and one attacker's model from {Support

Vector Machine, Logistics Regression, Ridge Regression, Neural Network, Ensemble Model}. The details for each model are introduced in Chapter 4.

6.4 Baseline Methods

We choose the most popular adversarial example attack, i.e., Fast Gradient Sign Attack attack(FGSM), as the baseline method [24]. Then we modify the basic FGSM attack with the transferability enhance techniques, i.e., ensemble-sample and ensemble-model, introduced in Chapter 3. Note that ensemble-model and ensemble-sample techniques have only been studied in the computer vision domain before while there are some different restrictions in the malware domain. We have covered how to adapt the two techniques to malware data in Chapter 3. We will compare the transferability performance results of these algorithms as follow:

6.4.1 FGSM attack

FGSM attack is one of the earliest yet effective adversarial example attacks. As our first baseline, we train the neural network with 3 layers and 100 neurons at each layer as the only victim model. Then we train different substitute models including SVM, LR, RR, and neuron networks with 2 layers and 100 neurons at each layer. Note that we have a modification restriction set mentioned in Chapter 3. We gradually increase the percentage of modifiable features in the modification restriction set ϵ from 0.01 to 0.50. For each substitute model and the victim model, we end the training process when the validation accuracy no longer improves.

6.4.1.1 Results

Table 6.1 shows the experimental results of the FGSM attack. As the modification restriction gets looser (ϵ gets higher), the attack transferability increases. This means that if the attacker is able to modify more features of a given malware, the generated adversarial example can successfully evade the malware detection system with a higher probability. However, the transferability tends to stop increasing after ϵ hits a certain level.

Besides ϵ, the transferability is also limited to the substitute model the attacker uses. As the table suggests, the NN(2,100) achieves the fastest-growing transferability performance while the other three machine learning models (SVM, LR, and RR) grow much slower to achieve their maximum transferability value. Using NN(2,100) as the substitute model gets much better transferability in the lower ϵ range (0.01 to 0.03 in this experiment). This is due to the very similar structure between the substitute model NN(2,100) and the victim model NN(3,100).

We also validate that previous research[17] which indicates that lower-complexity models (with stronger regularization) provide better surrogate models. As the ϵ increases, the other 3 linear models also achieve higher than 90% transferability due to their great generalization capability. SVM normally is designed to generate more complex decision boundaries[42] than the more complicated models such as LR and RR. It can be seen that LR and RR always outperform SVM in all ϵ settings. We cannot differentiate the transferability performance of LR and RR in this experiment. The only difference between the two substitute models is that RR has regularization in the objective function, therefore providing a simpler model. We tried different RR regularization values but cannot sway the experimental results. We guess it is because adding the regularization cannot affect the decision boundaries of the linear models easily due to the data and feature distribution we use.

Table 6.1: FGSM method attack transferability on neural network

Different Substitute models	Victim model(NN(3,100))								
	$\epsilon = 0.01$	$\epsilon = 0.02$	$\epsilon = 0.03$	$\epsilon = 0.04$	$\epsilon = 0.05$	$\epsilon = 0.06$	$\epsilon = 0.10$	$\epsilon = 0.20$	$\epsilon = 0.50$
SVM	4%	27%	73%	91%	91%	91%	91%	91%	91%
LR	5%	46%	84%	96%	96%	96%	96%	96%	96%
RR	5%	46%	84%	96%	96%	96%	96%	96%	96%
NN(2,100)	27%	68%	93%	94%	94%	94%	94%	94%	94%

6.4.2 Ensemble-Sample

As introduced in Chapter 3, ensemble-sample (ES) is one of the popular techniques in the computer vision domain. It uses multiple samples adapted from the original one to generate the AE for higher transferability. In this experiment, we design a similar experiment as the previous one 6.1 to examine whether it also works well in the malware detection domain.

Table 6.2: Attack performance of Ensemble-sample attack (FGSM) on deep neural network

Different Substitute models	Victim model(NN(3,100))								
	$\epsilon = 0.01$	$\epsilon = 0.02$	$\epsilon = 0.03$	$\epsilon = 0.04$	$\epsilon = 0.05$	$\epsilon = 0.06$	$\epsilon = 0.10$	$\epsilon = 0.20$	$\epsilon = 0.50$
SVM	4%	28%	74%	92%	92%	92%	92%	92%	92%
LR	5%	46%	84%	96%	96%	96%	96%	96%	96%
RR	5%	46%	84%	96%	96%	96%	96%	96%	96%
NN(2,100)	28%	68%	93%	94%	94%	94%	94%	94%	94%

6.4.2.1 Results

Table 6.2 shows the experimental results of FGSM method enhanced by ensemble-sample technique. Compared with Table 6.4.1, we can see that ensemble-sample only has minor improvements for a few settings in the Table such as SVM($\epsilon = 0.02$ to $\epsilon = 0.50$) and NN(2,200)($\epsilon = 0.01$). Ensemble-sample in this experiment performs relatively poorly in improving the adversarial example transferability.

6.4.3 Ensemble-Model

Ensemble-model (EM) is another major technique in the computer vision domain to improve AE transferability. In this experiment, we investigate how the ensemble-model method performs in the malware domain. Furthermore, we designed different victim models and the corresponding substitute models with the ensemble-model technique to evaluate the relationship between AE attack transferability and models complexity of the victim model and substitute model.

6.4.3.1 Results

Table 6.3: Attack performance of Ensemble-Model method (FGSM) on different victim models, Part I

Different Victim models	Corresponding attacker's EM	Modification restrictions					
		$\epsilon = 0.01$	$\epsilon = 0.02$	$\epsilon = 0.03$	$\epsilon = 0.04$	$\epsilon = 0.05$	$\epsilon = 0.06$
SVM	NN-LR-RR	8%	9%	16%	16%	17%	17%
LR	SVM-NN-RR	6%	9%	13%	21%	24%	24%
RR	SVM-NN-LR	3%	5%	11%	11%	12%	12%
NN	SVM-LR-RR	6%	44%	83%	95%	96%	96%

Table 6.4: Attack performance of Ensemble-Model method (FGSM) on different victim models, Part II

Different Victim models	Corresponding attacker's EM	Modification restrictions					
		$\epsilon = 0.01$	$\epsilon = 0.02$	$\epsilon = 0.03$	$\epsilon = 0.04$	$\epsilon = 0.05$	$\epsilon = 0.06$
SVM	SVM-NN-LR-RR	8%	9%	16%	17%	19%	19%
LR	SVM-NN-LR-RR	1%	5%	23%	30%	39%	40%
RR	SVM-NN-LR-RR	2%	3%	16%	21%	30%	30%
NN	SVM-NN-LR-RR	0%	1%	6%	17%	55%	55%

For the first objective, we examine the basic experiment as previous experiments did use NN(3,100) as the victim model and SVM-LR-RR as the ensemble substitute model. The

last row in Table 6.3 shows the results. Generating AEs on the substitute ensemble model could slightly surpass baseline FGSM and ensemble-sample FGSM as ϵ gets higher but perform even poorly at a small ϵ value (0.01 to 0.03).

To further investigate how the transferability is affected by model complexity, we ensembled one more model NN(3,100) (the same structure as the victim model) in the original SVM-LR-RR model as the new substitute model. The last row of Table 6.4 states the results that the transferability gets even lower in every ϵ setting. The reason is that the substitute model becomes more complicated after ensembling one NN. Generating AE on a complicated model but attacking a rather simple model would result in lower transferability. It is also consistent with the argument we made before in baseline FGSM experimental results.

We have more interesting findings by using different victim models and doing similar control experiments like the above one NN(3,100) as the substitute model. For the other 3 linear models as the victim model, after ensembling one more model with the same victim model structure, the upper bound of transferability performance would increase otherwise. We speculate the reason is that the substitute ensemble model compared with the simple linear victim model is already complicated(with NN included). The benefit of ensembling the similar model into the substitute model would be much more significant for enhancing AE transferability.

6.5 Experimental Results of Our Proposed Method

We show the experimental findings from our method as follows. In this experiment, we still use the same settings in the FGSM experiment 6.4.1 and the ensemble-sample experiment 6.4.2.

What is the relationship between ϵ and Transferability?

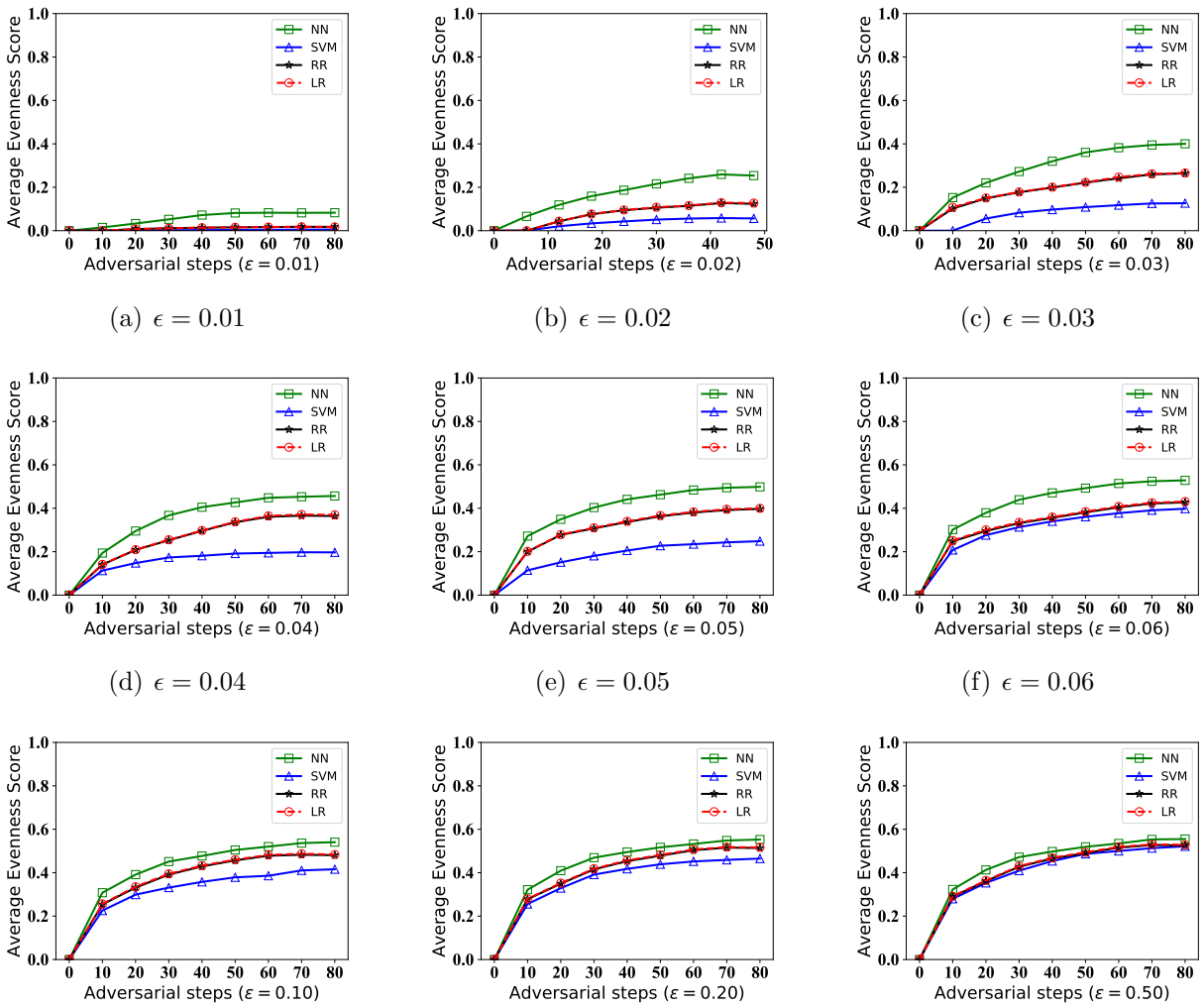
Table 6.5: Our method attack transferability on neural networks

Different Substitute models	Victim model(NN(3,100))								
	$\epsilon = 0.01$	$\epsilon = 0.02$	$\epsilon = 0.03$	$\epsilon = 0.04$	$\epsilon = 0.05$	$\epsilon = 0.06$	$\epsilon = 0.10$	$\epsilon = 0.20$	$\epsilon = 0.50$
SVM	0%	24%	47%	68%	76%	81%	90%	95%	95%
LR	1%	33%	58%	75%	88%	92%	97%	97%	98%
RR	1%	33%	58%	75%	88%	92%	97%	97%	98%
NN(2,100)	14%	43%	72%	89%	95%	96%	98%	98%	98%

As the results show, the transferability of our method performs poorly at small ϵ compared with previous baseline experiments. However, the results increase as ϵ increases finally will exceed the maximum value the previous base methods could get. When the ϵ is up to certain values according to different substitute models, the transferability performance will reach the peak, which excels FGSM baseline 2% to 4% and ensemble-model baseline (NN(2,100) as the substitute model) 2%. It is obvious that the more modifiable features, the higher evenness score our method could optimize, thus the higher transferability results the attacker could get.

Is the optimization problem solved?

To further validate how our method works and whether the evenness score gets optimized, we examine the average evenness score change as the adversarial steps increase in the gradient descent process. We plot the average evenness score over 100 testing samples when proceeding with optimization in different ϵ settings from 0.01 to 0.5 in Figure 6.1. When the ϵ in small values, average evenness scores remain constant as the adversarial steps increase since the optimization space is limited. As the ϵ gets large, the substitute model NN has the fastest growth rate in evenness score. The LR is slightly higher than RR while the SVM is the slowest one. The reason behind this phenomenon is that the NN has more nonlinear relationships in model space and could get overfitted more easily. Therefore, it has more potential to be optimized even in small ϵ. Also, since LR has no regularization penalty as in RR, it will get overfitted more easily than RR.



(a) $\epsilon = 0.01$ (b) $\epsilon = 0.02$ (c) $\epsilon = 0.03$

(d) $\epsilon = 0.04$ (e) $\epsilon = 0.05$ (f) $\epsilon = 0.06$

(g) $\epsilon = 0.10$ (h) $\epsilon = 0.20$ (i) $\epsilon = 0.5$

Figure 6.1: Evenness scores as the adversarial step increases.

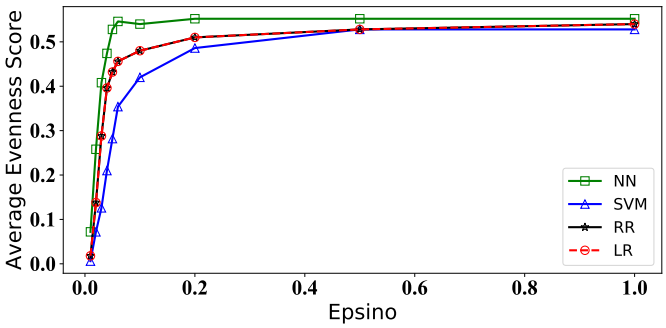


Figure 6.2: The best average evenness score as ϵ increases.

Even though the model non-linearity affects evenness score optimization space, it would be a minor factor if the ϵ value is large enough in this experiment. From Figure 6.1, we could observe that the evenness scores are very close in all models finally.

Figure 6.2 plots the upper limit of evenness score per sample averaged over 100 testing samples the algorithm could achieve as the ϵ increases. It also indicates the larger modifiable feature space would enlarge the evenness score optimization space.

What is the relationship between Evenness Score and Transferability?

Table 6.6: Correlation between transferability rate and evenness score. Each result is obtained from 100 testing samples.

		coefficients value	p-value
SVM	Pearson	0.68	1e-5
	Spearman Rank	0.65	1e-5
	Kendall's Tau	0.48	1e-5
LR	Pearson	0.75	1e-5
	Spearman Rank	0.72	1e-5
	Kendall's Tau	0.51	1e-5
RR	Pearson	0.75	1e-5
	Spearman Rank	0.72	1e-5
	Kendall's Tau	0.51	1e-5
NN(2,200)	Pearson	0.82	1e-5
	Spearman Rank	0.94	1e-5
	Kendall's Tau	0.81	1e-5

Here we investigate the relationship between Evenness Score and the transferability to further examine our intuition. In order to evaluate and investigate the statistical significance of the correlation between the evenness score and the transferability, we compute the associated correlation values with three metrics respectively: Pearson value, Spearman Rank value and Kendall's Tau value. They are shown in Table 6.6. We observed that the hypothesis that the evenness score and the transferability have no correlation failed. We obtained the test value from the transferability and the average evenness score of 100 testing samples in different ϵ settings (0.01 to 0.5). All the p-values are smaller than 10^{-5}, which confirms 99% statistical significance.

6.6 Discussions

In this section, we first set up several baseline experiments from baseline FGSM, and FGSM enhanced by ES and EM to evaluate how previous techniques work in improving malware AE transferability. Then we implement our evenness score-based algorithm in different modifiable feature settings and compare the results with the previous baseline experimental results.

Our algorithm outperforms ES and EM in large ϵ settings due to the larger evenness score optimization space. ES and EM could only slightly enhance the transferability compared with the baseline FGSM method in our experiments. We only use one dataset and did limited experiments in this thesis. Due to the complexity of malware feature properties and malware detection systems, we speculate that this property could be generalized in the malware domain. We could leave more general malware settings in future work.

From the attacker's side, when choosing the substitute model, a model with simple complexity would provide AEs with better transferability. If the modifiable feature set is guaranteed

large, evenness-score-based AE generation would have more advantages than baseline FGSM, ES, or EM. To further improve the transferability, the attacker could even combine these techniques together: ES, EM, and evenness score. We don't implement it here due to time and space limits and also leave it to future work.

To defend against the transfer AE attack, firstly the defender could use a relatively simple model as the victim model. It would result in AEs tending to be generated from a more complicated substitute model and make it harder to evade the victim one. But it could be the tradeoff between security in the AE transfer attack and malware detection performance. Since the attacker could enhance the transferability in a large modifiable feature set, the model owner could extract more malware features that are not allowed to modify as the model input vector to keep the number of modifiable features at a small value. Also, another direction is that the model owner could design smooth factors in the model training process to force the model to attribute each feature of each sample more evenly. In this way, the attacker could only have limited space to optimize the evenness score.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

This thesis first investigates the performance of AE attacks on machine learning-based malware detection systems. Specifically, we focus on the transferability property of AE in the malware domain. That is, from the attacker's side, how is the attack success rate of AEs generated from the substitute model performed on the targeted victim model. In this work, we consider the challenging and threatening scenario, the grey box attack, which indicates that the attacker would have limited knowledge about the victim model including the model architecture, the model parameters, and even the query results for AE samples.

Previous research only had limited works on AE transferability in the malware domain and showed poor performance on this property. To shed the light on it, we first investigate how previous transfer AE attacks work on malware classifiers. Furthermore, we survey the popular techniques in the computer vision domain to enhance the AE transferability and transplant and adapt them for malware data. Our results fill the gap and show that

previous techniques ES and EM could only have a minor improvement in transferability and only work in limited settings in our experiments.

We then propose a novel formulation for transfer AE attacks based on the contribution of each feature for an input sample towards the model prediction result. We define the evenness score and speculate that if an AE generated from the substitute model with a higher evenness score, it would have more chance to evade the target victim model. We propose the optimization problem to maximize the evenness score, and at the same time, successfully evade the substitute model. We adjust our optimization formulation to a simpler form according to Lagrangian relaxation and intend to implement the gradient descent to find the optimal solution.

We design comprehensive experiments to examine how our method performs. Experimental results indicate that the methodology we designed has excellent performance when the modifiable features set on malware data is large. In comparison, ES and EM would outperform our algorithm on a relative small modifiable features set. We then further analyze and speculate the reasons behind this. We explain and conclude in which scenarios our algorithm or baseline techniques would have better performance on transfer AE attacks. To validate our intuition, we assess the statistical significance of the relationship between the evenness score and the AE transferability on 3 different metrics: Pearson value, Spearman Rank value, and Kendall's Tau value. Results indicate a high correlation between the evenness score and attack transferability of AEs.

This thesis discusses the difference in the transfer AE attack between malware data and the computer vision domain in detail. We examine how previous transferability enhancement techniques work in the malware domain. We also put forward a new optimization-based method based on evenness score. In experiments, we further verify the high correlation between the evenness score and AE transferability. According to me, maximizing the evenness

score keeps the generated AE from overfitting to the substitute model. By perturbing the original malware sample more even, the generated AE could have a higher capability of generalization. It is normal to add smoother in the model to keep it from overfitting[14]. On the contrary, perturbing the AE more even is like trying to add the smoother on the AE to keep it from overfitting, from the attackers' side.

7.2 Future Work

Although comprehensive experiments have been done, there are still several potential extensions for this work.

7.2.1 Different adversary models

In this work, we assume the adversarial model is the grey-box attack model. However, there is still one strong assumption. We assume that the attacker uses the same training data the victim model uses. In the real scenario, it could be impossible. In future work, we should set different levels of the attackers' knowledge. The attacker could know a portion of the training data from 0% to 100%. Also, since different malware detection systems could use very different feature engineering methods, the AE transfer attack could be implemented in this situation where the attacker has no knowledge about the features used in the victim model.

7.2.2 Explainable machine learning

In our experiments, we directly use the gradient of each sample as the feature weights contributing to the prediction results. However, it could be more complicated and more precise

to define the contribution weights. Explainable machine learning[29, 38, 52] techniques use more information rather than pure gradient to locate and quantize the accurate feature attribution of each sample. However, since we use gradient descent to solve the optimization problem, using such techniques could also make the optimization problem more complicated and time-consuming. Heuristic algorithms could be the potential way for it.

7.2.3 Evenness score definition

In this work, we only use a very naive formulation to define how even the feature weights distribute. There could be a more appropriate definition for our problem. The definition of the evenness score should have 2 requirements: it should reflect the even level of feature weights and also be easy to be solved by gradient descent optimization.

7.2.4 Different datasets

Although our algorithm has been verified on the malware data, we expect that it should apply to more general scenarios such as image domain or network traffic data. Since the image domain does not have as many limitations as it is in malware, more experiments can be done to examine its effectiveness.

Bibliography

[1] Yousra Aafer, Wenliang Du, and Heng Yin. Droidapiminer: Mining api-level features for robust malware detection in android. In *International conference on security and privacy in communication systems*, pages 86–103. Springer, 2013.

[2] Shubair Abdulla and Altyeb Altaher. Intelligent approach for android malware detection. *KSII Transactions on Internet and Information Systems (TIIS)*, 9(8):2964–2983, 2015.

[3] Mohammed S Alam and Son T Vuong. Random forest classification for detecting android malware. In *2013 IEEE international conference on green computing and communications and IEEE Internet of Things and IEEE cyber, physical and social computing*, pages 663–669. IEEE, 2013.

[4] Shaikh Bushra Almin and Madhumita Chatterjee. A novel approach to detect android malware. *Procedia Computer Science*, 45:407–417, 2015.

[5] Hyrum S Anderson and Phil Roth. Ember: an open dataset for training static pe malware machine learning models. *arXiv preprint arXiv:1804.04637*, 2018.

[6] Hyrum S Anderson, Anant Kharkar, Bobby Filar, David Evans, and Phil Roth. Learning to evade static pe machine learning malware models via reinforcement learning. *arXiv preprint arXiv:1801.08917*, 2018.

[7] Anshul Arora, Shree Garg, and Sateesh K Peddoju. Malware detection using network traffic analysis in android based mobile devices. In *2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies*, pages 66–71. IEEE, 2014.

[8] 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.

[9] Battista Biggio, Igino 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.

[10] Iker Burguera, Urko Zurutuza, and Simin Nadjm-Tehrani. Crowdroid: behavior-based malware detection system for android. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, pages 15–26, 2011.

[11] Gerardo Canfora, Eric Medvet, Francesco Mercaldo, and Corrado Aaron Visaggio. Detecting android malware using sequences of system calls. In *Proceedings of the 3rd International Workshop on Software Development Lifecycle for Mobile*, pages 13–20, 2015.

[12] 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.

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

[14] Tianlong Chen, Zhenyu Zhang, Sijia Liu, Shiyu Chang, and Zhangyang Wang. Robust overfitting may be mitigated by properly learned smoothening. In *International Conference on Learning Representations*, 2020.

[15] 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.

[16] 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*, 16(4):711–724, 2017.

[17] Ambra Demontis, Marco Melis, Maura Pintor, Matthew Jagielski, Battista Biggio, Alina Oprea, Cristina Nita-Rotaru, and Fabio Roli. Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks. In *28th USENIX security symposium (USENIX security 19)*, pages 321–338, 2019.

[18] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li. Boosting adversarial attacks with momentum. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9185–9193, 2018.

[19] Yinpeng Dong, Tianyu Pang, Hang Su, and Jun Zhu. Evading defenses to transferable adversarial examples by translation-invariant attacks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4312–4321, 2019.

[20] Chun-I Fan, Han-Wei Hsiao, Chun-Han Chou, and Yi-Fan Tseng. Malware detection systems based on api log data mining. In *2015 IEEE 39th annual computer software and applications conference*, volume 3, pages 255–260. IEEE, 2015.

[21] Ali Feizollah, Nor Badrul Anuar, Rosli Salleh, Fairuz Amalina, Shahaboddin Shamshirband, et al. A study of machine learning classifiers for anomaly-based mobile botnet detection. *Malaysian Journal of Computer Science*, 26(4):251–265, 2013.

[22] Shree Garg, Sateesh K Peddoju, and Anil K Sarje. Network-based detection of android malicious apps. *International Journal of Information Security*, 16(4):385–400, 2017.

[23] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[24] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *ICLR'15*, 2015.

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

[26] Shifu Hou, Yanfang Ye, Yangqiu Song, and Melih Abdulhayoglu. Hindroid: An intelligent android malware detection system based on structured heterogeneous information network. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1507–1515, 2017.

[27] Weiwei Hu and Ying Tan. Black-box attacks against rnn based malware detection algorithms. In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[28] Yang Huan, Yu-qing ZHANG, Yu-pu HU, and Qi-xu LIU. Android malware detection method based on permission sequential pattern mining algorithm. *Journal on Communications*, 34(Z1):106, 2013.

[29] Beomsu Kim, Junghoon Seo, and Taegyun Jeon. Bridging adversarial robustness and gradient interpretability. *arXiv preprint arXiv:1903.11626*, 2019.

[30] Aleksander Kołcz and Choon Hui Teo. Feature weighting for improved classifier robustness. In *CEAS'09: sixth conference on email and anti-spam*. Citeseer, 2009.

[31] Bojan Kolosnjaji, Ambra Demontis, Battista Biggio, Davide Maiorca, Giorgio Giacinto, Claudia Eckert, and Fabio Roli. Adversarial malware binaries: Evading deep learning for malware detection in executables. In *2018 26th European signal processing conference (EUSIPCO)*, pages 533–537. IEEE, 2018.

[32] Felix Kreuk, Assi Barak, Shir Aviv-Reuven, Moran Baruch, Benny Pinkas, and Joseph Keshet. Deceiving end-to-end deep learning malware detectors using adversarial examples. *arXiv preprint arXiv:1802.04528*, 2018.

[33] 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.

[34] Dongfang Li, Zhaoguo Wang, and Yibo Xue. Deepdetector: Android malware detection using deep neural network. In *2018 International Conference on Advances in Computing and Communication Engineering (ICACCE)*, pages 184–188. IEEE, 2018.

[35] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. Delving into transferable adversarial examples and black-box attacks. *arXiv preprint arXiv:1611.02770*, 2016.

[36] Davide Maiorca, Francesco Mercaldo, Giorgio Giacinto, Corrado Aaron Visaggio, and Fabio Martinelli. R-packdroid: Api package-based characterization and detection of mobile ransomware. In *Proceedings of the symposium on applied computing*, pages 1718–1723, 2017.

[37] Marco Melis, Michele Scalas, Ambra Demontis, Davide Maiorca, Battista Biggio, Giorgio Giacinto, and Fabio Roli. Do gradient-based explanations tell anything about adversarial robustness to android malware? *International Journal of Machine Learning and Cybernetics*, 13(1):217–232, 2022.

[38] Grégoire Montavon, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus-Robert Müller. Explaining nonlinear classification decisions with deep taylor decomposition. *Pattern recognition*, 65:211–222, 2017.

[39] 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.

[40] 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 (EuroS&P)*, pages 372–387. IEEE, 2016.

[41] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pages 506–519, 2017.

[42] NLMM Pochet and JAK Suykens. Support vector machines versus logistic regression: improving prospective performance in clinical decision-making. *Ultrasound in Obstetrics and Gynecology: The Official Journal of the International Society of Ultrasound in Obstetrics and Gynecology*, 27(6):607–608, 2006.

[43] 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.

[44] Borja Sanz, Igor Santos, Carlos Laorden, Xabier Ugarte-Pedrero, Pablo Garcia Bringas, and Gonzalo Álvarez. Puma: Permission usage to detect malware in android. In *International Joint Conference CISIS'12-ICEUTE 12-SOCO 12 Special Sessions*, pages 289–298. Springer, 2013.

[45] Bhaskar Pratim Sarma, Ninghui Li, Chris Gates, Rahul Potharaju, Cristina Nita-Rotaru, and Ian Molloy. Android permissions: a perspective combining risks and benefits. In *Proceedings of the 17th ACM symposium on Access Control Models and Technologies*, pages 13–22, 2012.

[46] Asaf Shabtai, Yuval Fledel, Yuval Elovici, and Yuval Shahar. Using the kbta method for inferring computer and network security alerts from time-stamped, raw system metrics. *Journal in computer virology*, 6(3):239–259, 2010.

[47] Asaf Shabtai, Uri Kanonov, Yuval Elovici, Chanan Glezer, and Yael Weiss. "andromaly": a behavioral malware detection framework for android devices. *Journal of Intelligent Information Systems*, 38(1):161–190, 2012.

[48] Avanti Shrikumar, Peyton Greenside, Anna Shcherbina, and Anshul Kundaje. Not just a black box: Learning important features through propagating activation differences. *arXiv preprint arXiv:1605.01713*, 2016.

[49] Wei Song, Xuezixiang Li, Sadia Afroz, Deepali Garg, Dmitry Kuznetsov, and Heng Yin. Mab-malware: A reinforcement learning framework for attacking static malware classifiers. *arXiv preprint arXiv:2003.03100*, 2020.

[50] Jack W Stokes, De Wang, Mady Marinescu, Marc Marino, and Brian Bussone. Attack

and defense of dynamic analysis-based, adversarial neural malware detection models. In *MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM)*, pages 1–8. IEEE, 2018.

[51] Octavian Suciu, 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.

[52] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *International conference on machine learning*, pages 3319–3328. PMLR, 2017.

[53] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014. URL http://arxiv.org/abs/1312.6199.

[54] Guanhong Tao, Zibin Zheng, Ziying Guo, and Michael R Lyu. Malpat: Mining patterns of malicious and benign android apps via permission-related apis. *IEEE Transactions on Reliability*, 67(1):355–369, 2017.

[55] Ke Tian, Danfeng Yao, Barbara G Ryder, Gang Tan, and Guojun Peng. Detection of repackaged android malware with code-heterogeneity features. *IEEE Transactions on Dependable and Secure Computing*, 17(1):64–77, 2017.

[56] Florian Tramèr, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. The space of transferable adversarial examples. *arXiv preprint arXiv:1704.03453*, 2017.

[57] Wei Wang, Xing Wang, Dawei Feng, Jiqiang Liu, Zhen Han, and Xiangliang Zhang. Exploring permission-induced risk in android applications for malicious application de-

tection. *IEEE Transactions on Information Forensics and Security*, 9(11):1869–1882, 2014.

[58] Lei Wu, Zhanxing Zhu, Cheng Tai, et al. Understanding and enhancing the transferability of adversarial examples. *arXiv preprint arXiv:1802.09707*, 2018.

[59] Weibin Wu, Yuxin Su, Michael R Lyu, and Irwin King. Improving the transferability of adversarial samples with adversarial transformations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9024–9033, 2021.

[60] Wen-Chieh Wu and Shih-Hao Hung. Droiddolphin: a dynamic android malware detection framework using big data and machine learning. In *Proceedings of the 2014 Conference on Research in Adaptive and Convergent Systems*, pages 247–252, 2014.

[61] Xi Xiao, Xianni Xiao, Yong Jiang, Xuejiao Liu, and Runguo Ye. Identifying android malware with system call co-occurrence matrices. *Transactions on Emerging Telecommunications Technologies*, 27(5):675–684, 2016.

[62] Cihang Xie, Zhishuai Zhang, Yuyin Zhou, Song Bai, Jianyu Wang, Zhou Ren, and Alan L Yuille. Improving transferability of adversarial examples with input diversity. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2730–2739, 2019.

[63] Shiting Xu, Xinyu Ma, Yuandong Liu, and Qiang Sheng. Malicious application dynamic detection in real-time api analysis. In *2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 788–794. IEEE, 2016.

[64] Weilin Xu, Yanjun Qi, and David Evans. Automatically evading classifiers. In *Proceedings of the 2016 network and distributed systems symposium*, volume 10, 2016.

[65] Hui-Juan Zhu, Zhu-Hong You, Ze-Xuan Zhu, Wei-Lei Shi, Xing Chen, and Li Cheng. Droiddet: effective and robust detection of android malware using static analysis along with rotation forest model. *Neurocomputing*, 272:638–646, 2018.