A PROPOSED DEFENSE FRAMEWORK AGAINST ADVERSARIAL MACHINE LEARNING ATTACKS USING HONEYPOTS.

by

Fadi Younis

B.Sc. Ryerson University, Toronto (ON), Canada, 2009

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Science

in the Program of

Computer Science

Toronto, Ontario, Canada, 2018

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my thesis may be made electronically available to the public for the purpose of scholarly research only.

Abstract

A proposed Defense Framework Against Adversarial Machine Learning Attacks Using Honeypots.

Fadi Younis
Master of Science, Computer Science
Ryerson University, 2018

This is my wonderful abstract. I really like my abstract because it's so pretty. I've made sure it's less than 150 words if I'm a M.Sc. student and less than 350 words if I'm a Ph.D. student. I've not included any graphs, tables or references in it. If I really had to use symbols in my abstract I'd be sure to also explain right here what they stand for because I'm such a good student and I know all the thesis rules.

Acknowledgements

Many people have contributed to my work here at Ryerson University. First I thank my supervisor Dr. Ali Miri for guiding my research, as well as providing many helpful suggestions throughout my time here. I would like to acknowledge the Department of Computer Science for providing me funding while I did my research.

To

To my friends, My dear family and and wonderful colleagues, Without whom none of my success would be possible.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this chapter, we introduce the thesis problem setting in which Adversarial Examples occur and maliciously influence the prediction model (**Section 1.1**). Then, we provide an overview of the thesis problem subject matter (**Section 1.2**), the motivation for solving the problem follows after (**Section 1.3**). We then outline the goal(s) we hope to reach with our solution, by the end of this thesis (**Section 1.4**). Finally, we end the chapter with an outline for the remainder of this thesis (**Section 1.5**).

1.1 Setting

Machine Learning, as we know it, is exploding in development and demand, with utilization in critical applications, services and domains, not confined to one area of industry but several. With each day, more applications are harnessing the suitability of Machine Learning. To meet the ever increasing demand that this forefront is witnessing, tech giants such as Amazon, Google, Uber, Netflix, Microsoft and many others are providing their Machine Learning adjuncted products and services in the form of an online cloud service, otherwise known as *Machine-Learning-As-A-Service* (MLaaS) [19]. While the need for easily accessible Machine Learning tools is becoming readily available, the desire to personally customize and build these services from the ground up is actually decreasing and less relied upon. This is because users do not wish to spend countless hours training, testing and fine-tuning their machine learning models, they simply want to readily use them. While some users still prefer to ordain and control how their models are constructed and deployed, companies have undergone the effort to hide the internal complex mechanisms from most of their indolent users, and package them in non-transparent and easily customized services. Essentially, they provide their services

in the form of a *Black-Box* [12] [16]. This opaque system container accepts some input and produces an output, but where the internal details of the model are hidden. However, like any application deployed, we cannot assume it is sited in a safe environment. There are security flaws and vulnerabilities in every man-made system and the container holding a MLaaS is no exception to the assumption. These weaknesses introduce a susceptibility to malicious attack(s), which is expected and almost always the case. Like any regular user, an adversarial attacker also does not have access to the internal components of the system, this knowledge might give a sense of security, and that the *Black-Box* system is secure. But as well see, this sense of security is only temporary, and only the beginning of the dangers to follow.

1.2 The Problem at a Glance

As we learned in the introduction of this chapter, Machine Learning Adversarial threats exist and are lurking close-by. The threat transpires when an attacker misleads and confuses the prediction model inside the cloud computing application offering the MLaaS, and allow malicious activities to go undetected [16]. This drives up the rate of false negatives, violating model integrity. These masqueraded inputs - called Adversarial Examples [12], represent one of the recent threats to cloud services providing *Machine Learning as a Service* (MLaaS). These nonlinear inputs look familiar to the inputs normally accepted by a linearly designed classifier, but only appear that way. Theyre maliciously crafted to exploit blind spots in the classifier boundary space, to mislead and confuse the learning mechanism in the classifier, to compromise the model integrity, post training. Most defenses aim at strengthening the classifiers discriminator function, by training it on malicious input ahead of time to make it robust. Defense methods, such as *Regularization and Adversarial Training* have proven unsuccessful. The latter method, and others like it, alone, cannot be relied upon since they do not generalize well on new adversarial inputs. This is evidently true in the case of a Black-Box (blind model) setting, where the adversary has access to only input and output labels, as mentioned. Our aim is to develop an adversarial defense framework to act as a secondary level of prevention to curb adversarial examples from corrupting the classifier, by deceiving the attacker.

1.3 Motivation

Due to the exploding demand that machine learning is witnessing, the risk of adversarial threats has increased as well. For example, an attacker can maliciously fool an *Artifi-*

cial Neural Network (ANN) classifier into allowing malicious activities to go undetected, without direct influence on the classifier itself (ref). These masqueraded inputs - called Adversarial Examples, represent one of the recent threats to cloud services providing Machine Learning as a Service (MLaaS). Theyre maliciously crafted to exploit blind spots in the classifier boundary space, to mislead and confuse the learning mechanism in the classifier, post model training, to compromise the integrity of the model. As a result there has been an increased interest in defense techniques to combat them.

Our challenge here lies in constructing an adversarial defense technique capable of deceiving the intrusive attacker and lure him away from the black-box target model. For purposes of our approach, we have decided to primarily use *Adversarial Honey-Tokens*, which act as fictional digital breadcrumbs designed to lure the attacker and detectable using network sniffers used by the attacker. It is possible to generate a unique token for each item to deceive the attacker and track his abuse, however each token must be designed, generated and strategically embedded into the system to misinform and fool the adversary.

previous research has aimed at strengthening the classifiers discriminator function, by training it on malicious input to make it robust. Defense methods, such as *Regularization and Adversarial Training* have proven unsuccessful. The latter method has been criticized because it cannot be relied upon since they do not generalize well on new adversarial inputs [20]. This is evidently true in the case of a Black-Box (blind model) setting, where the adversary has access to only input and output labels. We believe it is necessary to develop an adversarial defense framework to act as a secondary level of protection to prevent adversarial examples from corrupting the classifier, by deceiving the attacker. The majority of our is designing a distributed network of High-Interaction Honeypots as an open target for adversaries, these honeypot nodes act as sandboxes to contain the decoy neural network, collect valuable data and insight into adversarial attacks. We believe this might deter adversaries from attacking the target model. Other adversarial example defenses can also benefit and utilize this framework as an appendage in their techniques. Unlike other proposed defense models in literature, our model prevents the attacker from interacting directly with the target.

We designed our defense framework to deceive the adversary in 3 steps, occurring sequential order of each other. The information collected from the attacker's interaction with decoy model could then potentially be used to learn from the attacker, re-train and robustify the deep learning model in future training iterations. Our defense approach is motivated by trying to answer the following question *"is there a possible way to fool the attacker and prevent him from learning behavior of the model"*. At its core, our intention

is to devise a defense technique to both fool and prevent the attacker from interacting with the model.

1.4 Thesis Goals

Our thesis is as follows:

"Given a deep learning model within a Black-Box setting and a intrusive attacker with an adversarial input capable of corrupting the model and misclassifying its labels there exists a defense framework to fool and deter the attacker's attempts. If building such a defense framework is possible then there exists a way to prevent the attacker from learning the model's behavior and corrupting it."

The purpose of this work is to investigative the thesis above and build a an appropriate defense framework that implements it. Due to time and resource constraints, we limit our objectives to the following:

- Propose an adversarial defense approach, that will act as a secondary level of defense to reinforce existing adversarial defenses. It aims to: 1) preventing the attacker from correctly learning the classifier labels, and approximating the correct architecture of the Black-Box, 2) luring the attacker away from the model towards a decoy model, and 3) create an infeasible computational work for the adversary, with no functional use.
- Evaluate performance of the proposed defense method in the thesis, its strengths, weaknesses, limitations and benefits.
- Provide a detailed architecture and implementation of the *Adversarial Honey-Tokens*, their design, features, usage, deployment and benefits.

1.5 Overview

This thesis has 7 chapters. It is divided as follows:

- **Chapter 1 -** gives a brief introduction to the problem at hand and its setting. This chapter also gives an overview of the thesis goals, and a breakdown of the outline.

- **Chapter 2 -** introduces the problem, and the motivation for solving it. It also introduces relevant concepts, such as *Adversarial Machine Learning*, *Transferability*, *Deep Neural Networks*, *Black-Box Systems*, and *Honeypots*.
- **Chapter 3 -** gives a summary and critical evaluation of the related work(s) authored by other researchers on the topic of adversarial black box attacks and proposed defenses.
- **Chapter 4 -** outlines the design and architecture of the defense approach. It also gives insight into the approach's setup, environment and limitations. It also shows how Honeypots can be used to curb adversarial attacks from influencing the model.
- **Chapter 5 -** details the approach set-up, adversarial environment, limitations and evaluation criteria.
- **Chapter 6 -** details implementation of the *Adversarial Honey-Tokens*, its features, usage, deployment and benefits.
- **Chapter 7 -** summarizes the thesis, gives an overview of the contribution(s) made and suggests future research directions.

Chapter 2

Background

The aim of this Chapter is to introduce the main problems and challenges involved in defenses against *Adversarial Examples*. We also formulate important concepts so that they can be used to understand the upcoming chapters. We begin by throughly explaining concepts that will be important for the rest of this thesis.

2.1 Deep Learning and Security

2.1.1 Deep Neural Networks

2.1.2 Deep Learning Vulnerabilities

2.1.3 Adversarial Training of Deep Learning Networks

2.1.4 Attacks on Deep Learning Networks

2.2 Adversarial Examples

2.3 Adversarial Example Attacks

In this section, we provide a thorough definition of adversarial examples. Also, relevant in this context, we describe the properties that give adversarial examples their potency, which has earned them their place as one of the most notorious threats to machine learning classifiers. However, We can't discuss adversarial examples without delving into how the techniques used to generated them, as well as the impact they have on deep neural networks. We also explore the known defenses techniques against adversarial examples.

2.3.1 Adversarial Examples Definition

As mentioned, machine learning models are vulnerable to adversarial attacks, that seek to destabilize the neural network and jeopardize its security and integrity. From what we learned in [24], these attacks can either occur during the training phases as a *poisoning attack* or testing phase as an *evasive attack* on the classification model. In a test-time attack scenario, the attacker actively attempts to circumvent and *evade* the learning process achieved by training the model. This is done by inserting inputs that exploit *blind spots* in the model, that remain undetected. These disruptive anomalies are known as *Adversarial Examples*.

Adversarial examples are (slightly) perturbed versions of inputs to classifiers. They are maliciously designed to have the same appearance as regular input, from a human's point of view, at least. These masqueraded inputs are designed to confuse, mislead, and force the classifier to output the wrong label [9], violating the integrity of the model. These examples can be best thought of as "glitches" that can fool the machine learning model. These glitches are difficult to detect and exploitable, if left unattended. To understand adversarial examples, consider the following: a well crafted adversarial input sample x to the classifier *A(.)*, the input x was is first originally correctly classified. Then given the same classification model *A(.)* with input x, so that *C(x)*= ℓ, we say $x^{'}$ is an adversarial example if *C(*$x^{'}$*)*$\neq$ ℓ. For an illustration of what adversarial examples are, see the figure 2.1 below for a better understanding of what is constructed by an adversarial example [15]. Classification models are considered *robust* if they are not affected by adversarial examples.

2.3.2 Adversarial Example Properties

In the beginning of the previous section (section 2.3.1) we mentioned that adversarial examples $x^{'}$ possess an appearance similar or *close* to the original input samples x. This degree of closeness could be L_2, which is the *Euclidean Distance* between two pixels in an image, L_{infity}, which is the absolute change made to a pixel, or L_1 , which is the total number of pixel changes made to the image [4]. If the measure of distortion in any of the previous metrics of closeness, then those images must be visually similar to each other. In our thesis work, we are assuming the attacker is using L_{infity} to measure the absolute change made to a pixel. Another interesting property, which our thesis topic is heavily based and which we are trying with our defense approach is the *transferability property*. This property states that given two models $F(\cdot)$ and $G(\cdot)$. an adversarial example on F will transfer to G, even if they're trained with two different neural net architecture, or

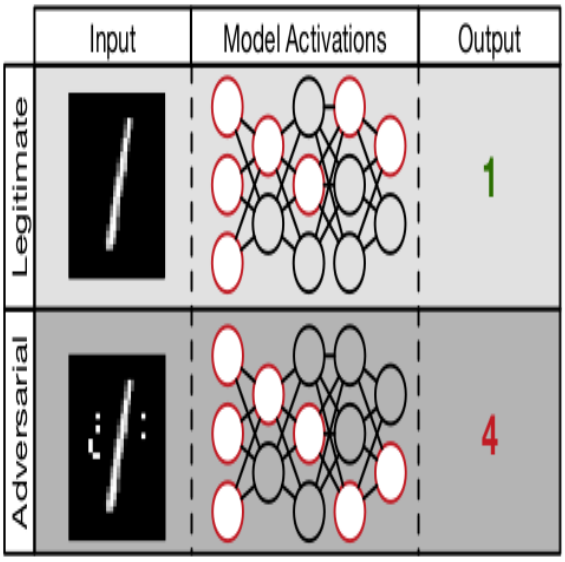

Figure 2.1: An Adversarial Example is generated by modifying a legitimate input sample, in such a way which would cause to the classifier to mislabel it, where as a human wouldn't notice a difference.

training sets [4]. This intriguing property of transferability enables the attacked model to generalize on the adversarial example, and allow both models $F(\cdot)$ and $G(\cdot)$ to assign the adversarial example to the same class. In order to understand why this property of transferability is possible, it is important to understand why it occurs and where these adversarial examples originate from. Work in [7] shows that adversarial examples are not scattered out randomly but occur in large and continuous regions in the decision space. The latter property might explain why transferability is possible in this regard. The authors in [7] argue that the higher dimensionality space of these two models, the more likely that the subspaces in these two models will intersect significant, and an adversarial example will be found that can be assigned to the same class label in both models. Furthermore, the authors in [23] argue that due to the shared dimensionality property, the decision boundary of any two models that have an adversarial example transfer from model $F(\cdot)$ and $G(\cdot)$ must very close to each other.

2.3.3 Generating Adversarial Examples

There are several techniques and methods to generating adversarial examples, used in experiments, such as those in [12] [16]. We refer to these product of these methods as adversarial examples. In order for us to understand how these minatory objects are

generated, we must first understand the individual components needs in their creation. Consider the following notation [12]:

X - a clean input (untampered) for some dataset *D*, typically a 3-D tensor (width × height × depth). Generally, the image pixel values range between *0* and *255*.

y(true) - the corresponding true label for input ***X***.

J(X, y) - the cross-entropy cost function used to train the model, given an an input image *X*, which we wish to maximize the loss function ***J()*** for.

ϵ***, n*** - the perturbation factors added to influence the model *f( )* to create the adversarial example x^*. ϵ represents the magnitude of perturbation in the image, with respect to the metric norm of closeness (L_1, L_2, or $L_{infinity}$). We want to keep ϵ *small* to a degree, to remain undetected by the observer.

As mentioned, there are several methods to generate adversarial examples. We won't delve into depth for each method, however we will focus most of our efforts exploring the *Jacobian-based Salience Map Approach* (JSMA), which the adversary in thesis uses in his attack.

2.3.4 Impact of Adversarial Examples on Deep Neural Networks

-how do they effect neural networks

2.3.5 Adversarial Examples Defenses

-list them

2.4 Transferability and Black-Box Learning Systems

The section focuses on the concept of Black-Box Learning Systems. We will offer a detailed definition of what a *Black-Box* Threat Model is, as well as how it contrasts from the *Blind* Threat model. We explore the functionality of of the Black-box Attack approach, as well the hypothesis responsible for allowing Black-box attacks to occur - *Adversarial Transferability* . This section also offers insight into how Adversarial Transferability is utilized to exploit and launch Black-Box attacks on classification models. Let us first introduce the concept of Adversarial Transferability.

2.4.1 Adversarial Transferability

According to Papernot and Goodfellow, who are known as the proponents of *Adversarial Transferability* in [23], the hypothesis of *Adversarial Transferability* is formulated as the following:

"If two models achieve low error for some task while also exhibiting low robustness to adversarial examples, adversarial examples crafted on one model transfer to the other."

In simple terms, the idea behind *Adversarial Transferability* is that for an input sample x, the adversarial examples A generated to confuse and mislead one model m can be *transfered* and used to confuse other models $\hat{m}$ that are of homogeneous or even heterogeneous classifier architectures. This mysterious phenomena is mainly due to the property commonly shared by most, if not all machine learning classifiers, which states that predictions made by these models vary smoothly around the input samples making them prime candidates for adversarial examples [9]. Also, it is also worth noting these perturbed samples, referred to here as *adversarial examples*, do not exist in the decision space as a mere coincidence. But according to [7], they occur within large regions of the classification model decision space. Here, dimensionality of the data is a crucial factor associated with the transferability of adversarial examples. The higher the dimensionality of the training data, the more likely that the subspaces will intersect significantly, guaranteeing the transfer of samples between the two subspaces. [7].

According to the above hypothesis, transferability holds true between two models, as long as both models share a similar purpose or task [15]. Knowing this, an attacker can leverage the property of *transferability* to launch an preemptive attack, by training a local substitute classifier B on training data that the chosen target classifier A was trained on, with a dataset x of similar dimensions and content, producing adversarial examples

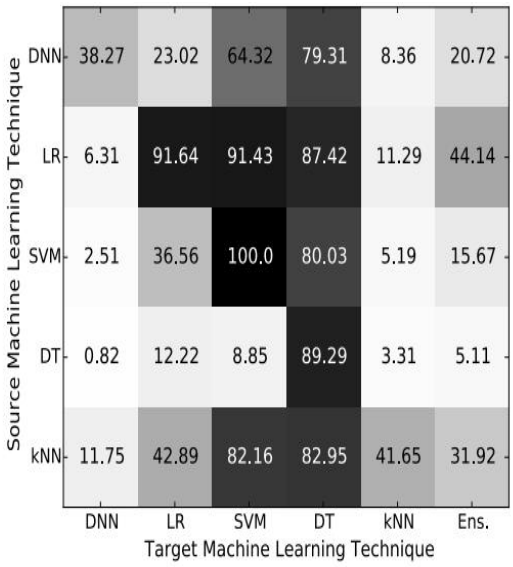

Figure 2.2: cross-technique Transferability matrix: cell (i,j)is the percentage of adversarial samples crafted to mislead a classifier learned using machine learning technique i that are misclassified by a classifier trained with technique j.

E. It is also worth noting that the success rate of transferability varies depending on the type of target classifier the examples E are being transferred to. The figure below 2.2 [9] illustrates the transferability matrix for adversarial examples generated with classifier of prediction technique i and transfered to a target classifier trained with technique j. As it can be seen in 2.2 the success rate of transferability is higher in some classification models such as *Support-Vector-Machines* (SVM) and *Linear Regression* (LR), but not others. The latter might be associated with the purity of data being used in generating the examples transferred [9].

These modified examples can then be transferred to the target classifier. Hence, the same perturbations that influence model A also effect model B. Knowing that the above hypothesis is true in the general case, Papernot used this very same concept to attack learning systems using adversarial examples generated and transferred from a substitute classifier in [16]. This transfer property is an anomaly, and creates an obstacle in the face of deploying and securing machine learning services on the cloud, enabling exploitations and ultimately attacks on Black-Box systems [23], as we'll see in the coming sections.

To measure the transferability of adversarial examples between two model m and $\hat{m}$, we use two points of measurement, which are: 1) *transferability rate* , 2) *success rate*. These two relationships of semblance are used to benchmark the *transferability* of the adversarial samples transfered from the substitute model back to the original model [16].

The success rate refers to the portion of perturbations that will be misclassified by the substitute model F, while the transferability refers to those same perturbation samples that will misclassified by the target model, when transferred form the substitute model.

2.4.2 Black-Box Threat Model

To understand what a *Black-Box Threat Model* is, we must first understand what the term *Black-Box* means. A Black-Box is essentially a system that can be construed in terms of inputs x and outputs y, with the internal mechanisms of the system $f(x) = y$ remains invisible. The functionality of the Black-Box can only be understood by observation. Figure below 2.3 illustrates a basic Black-Box system.

The Black-Box Threat Model is by extension a Black-Box system. In our thesis, we

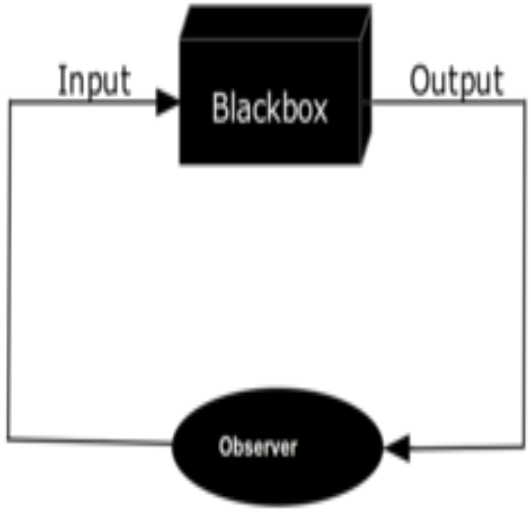

Figure 2.3: A Simple Black-Box System

are attempting to prevent the attacker from polluting the target classifier by blocking transferability and access to the target model to change the prediction on the class label. Here, we consider the adversary to be *weak*, as in he can only observe the inputs inserted and outputs produced. The adversary possess very little, if no knowledge at all of the classifier architecture, structure, number or type of hyper-parameters, activation function, node weights, etc. Such an environment is considered to be a *black-box system* and the type of attacks are called *black-box attacks*. The adversary needs not know the internal details of the system to exploit and compromise it [16].

Generally in order to attack the model, in Black-Box Learning setting, the adversary

attempts to generate adversarial examples, which are then transferred from the substitute classifier to the target classifier, in an effort to successfully distort the classification of the output labels [9]. The intension of the attacker is to train a substitute classifier in a way that is to mimic or simulate the decision space of the target classifier. For the later purpose, the attacker continuously updates the classifier and query the target classifier (Oracle) for labels to train the substitute model, craft adversarial examples and attack the black-box target classifier.

Generally, the model being targeted is a multi-class classifier system, otherwise known as the *Oracle O*. The Oracle referred to in this case represents the only capability which the attacker possesses, which is to query the Oracle for the label $O\vec{x}$ for any input $\vec{x}$ querying the Oracle O. This is the only capability available to the attacker, as in the Black-Box model no access to the Oracle internal details is possible [16].

The goal of the adversary is to produced a *perturbed* version of any input $\vec{x}$, known as an *adversarial sample* after modification, denoted $\vec{x}*$, this represents an attack on the integrity of the classification model (oracle) [16]. What the adversary attempts to do is solve the following optimization problem to generate the adversarial samples, as seen in Figure 2.9 below:

$$\vec{x^*} = \vec{x} + \arg\min\{\vec{z} : \tilde{O}(\vec{x} + \vec{z}) \neq \tilde{O}(\vec{x})\} = \vec{x} + \delta_{\vec{x}}$$

Figure 2.4: Adversarial Example Optimization Problem

The adversary must able to solve this optimization problem by adding a perturbation at an appropriate rate with $\delta\vec{x}$, to avoid human detection. The rate must chosen in such a way with the least perturbation possible to influence the classifier, as well remain undetected by a human [16]. This is considered a hard optimization problem, since finding a minimal value to $\delta\vec{x}$ is no trivial task, as mentioned in the above section. Further more, removing knowledge of the architecture and training data makes it difficult to find a perturbation, where $O\{\vec{x} + \delta\vec{x}\} = O\{\vec{x}\}$ [16].

2.4.3 Black-box Threat Model Vs. Blind Threat Model

Although our research mainly focuses on curbing adversarial attacks on black-box learning systems, it is also worth mentioning that other threat models exist. The different ones differ depending on how the adversary plans to query the substitute model and attack the target system. We have already been introduced to the first system, as mentioned above, the *Black-Box Threat Model*. While the more constrained second one is the *Blind Model*.

Opposite to the Black-Box Model, the Blind Model possesses a very limited (small) set of exposed knowledge to the attacker. This limited access applies to the labeled training data and its distribution. Unlike the *Black-Box Model*, the adversary is blind to the target system he is attacking [9], this means virtually no access. The internal details of the classifier are essentially shielded from the attacker. However, both Threat Models share some commonalities between them, for instance, Both models involve the attacker training a substitute classifier and use the perturbations generated to attack the target model [9]. They share their innate vulnerability to adversarial attacks, due to *Adversarial Transferability*, whose effect is more potent in the Black-Box Model than its adjacent Blind Model [9]. Also, the threat model being blind does not prevent it from being exposed to external attacks. Another interesting shared trait is the lack of robustness that the adversarial defense known as *distillation* [17] has inside both models. It was shown that in both models the attacker can evade the effect of its defense method of distillation by adversarial feature blocking which the defense depends on to thwart attacks [3].

2.4.4 Transferability in Black-Box Attacks

With Adversarial Transferability in mind, the adversary can build a substitute model F model with synthetic labels collected by observing the *Oracle* O. The attacker can build a substitute model F from what he learns from O. The attacker will now craft adversarial samples that will be misclassified by the substitute model F [17]. Now that the attacker has the knowledge of the internal architecture of F, he can use it to construct adversarial examples using one of the method described in section 2.3.3. For as long as adversarial transferability holds between F and O. adversarial examples misclassified by F will be misclassified by O.

2.4.5 Black-Box Attack Approach

As mentioned in the section 2.5.2, the adversary wishes to compromise the integrity of the classification model by querying the labels provided by the *Oracle* O for the input $\vec{x}$. According to [**?**]he adversary's plan is use the labels, collected by observing the *Oracle* O to generate a substitute model. The adversarial goal of finding a minimal perturbation to misclassify a targeted classifier model is a difficult problem, which happens to be non-convex, where multiple solutions exist for the global minimum of the optimization problem [16]. What the adversary can do is attempt mimic or approximate the *Oracle* O model architecture, but this requires internal knowledge of the classifier internal structure, which is not possible considering that inaccessibility under a Black-Box model

scenario [16]. Another benefit to this approach is that fact that most machine learning models require large and expensive training datasets. This makes incredibly difficult for the attacker to attack a system in such a environment [16]. The Black-Box attack strategy consists of the following steps:

Substitute Model Training:

Here, the attacker queries the *Oracle O* with synthetic inputs generated using one of the adversarial samples selected by one of the adversarial training algorithms to build a substitute model F, which will be used to misclassifying the target model due to the *adversarial transferability* property [16]. The notion of create a substitute model F is considered challenging due to two main reasons: 1) selecting an architecture F is difficult since we have no knowledge of the *Oracle O* model, 2) the number of queries to the *Oracle O* is limited to remain undetected [16]. The authors in [16] emphasis that their Black-Box approach is not meant to increase substitute model F accuracy, but to approximate the decision boundaries3s as best as possible, with few labels [16].

Substitute Architecture:

according to the authors in [16], this step is considered the most limiting factor when it comes to attacks on Black-Box Learning systems. This is due to the notion that without knowledge of the target model architecture, the attacker knows very little about how the system processes input (text, images, or media) and produces output (label or probability vector). One potential way, suggested by the authors, to select an appropriate architecture for the substitute model is to simply explore and try different variations of the substitute architectures and select one that yields that highest level of success [16].

Generating Synthetic Architecture:

another complication in the path of building a successful Black-Box attack is the dataset used to train the substitute model F. We could potentially request an infinite number of queries to get the oracle's output $O\vec{x}$ for an input sample $\vec{x}$ [16]. However, this method, although effective in the sense to create a copy of the Oracle is actually not methodical, this is due to the excess *Oracle O* querying. Creating a large number of queries attracts attention from the defender and makes the adversarial attempts to detect, and ultimately deflect [16].

Algorithm 1 - Substitute DNN Training: for oracle $\tilde{O}$, a maximum number max_ρ of substitute training epochs, a substitute architecture F, and an initial training set S_0.

Input: $\tilde{O}$, max_ρ, S_0, λ

1: Define architecture F
2: **for** $\rho \in 0 \,..\, max_\rho - 1$ **do**
3: *// Label the substitute training set*
4: $D \leftarrow \left\{(\vec{x}, \tilde{O}(\vec{x})) : \vec{x} \in S_\rho\right\}$
5: *// Train F on D to evaluate parameters θ_F*
6: $\theta_F \leftarrow \text{train}(F, D)$
7: *// Perform Jacobian-based dataset augmentation*
8: $S_{\rho+1} \leftarrow \{\vec{x} + \lambda \cdot \text{sgn}(J_F[\tilde{O}(\vec{x})]) : \vec{x} \in S_\rho\} \cup S_\rho$
9: **end for**
10: **return** θ_F

Figure 2.5: Substitute Model Training

Substitute Model Training:

here, we describe the five step algorithm procedure described in [16]. The algorithm is outlined in full below in figure 2.5 below. For a better understanding of how this algorithm is used to train substitute model, see below for a brief description of the steps involved [16]:

Initial Collection (step 1): the adversary collects a small set of input samples that representative of the *Oracle O* input domain. These inputs do not need to necessarily come from the same statistical distribution.

Architecture Selection (step 2): the adversary selects an architecture to be trained as the substitute model *F*.

Substitute Training the adversary selects more appropriate substitute models Fp by repeating the following steps:

Labeling (step 3): the labeled output $\tilde{O}\vec{x}$ is collected by querying the *Oracle O* . These labels are then used to compose the substitute model training set Sp to train the substitute model *F*.

Training (step 4): the adversary trains the substitute architecture model selected in step 2 above and trained using known adversarial training techniques, using labeled data Sp collected from step 3.

Augmentation (step 5): the authors augmentation technique in [16] is used on the training set Sp in order to produce a much larger training set S$p+1$ with more training data points. step3 and step4 are repeated for the augmented dataset. Step 3 is repeated several times to increases the substitute model F accuracy and mimic the decision boundaries of the *Oracle O*. Here, λ is the parameter of augmentation, which represents the step taken in the direction to augment from Sp to S$p+1$.

2.4.6 Defense Strategies Against Black-Box Attacks

According to [16], there are two main methodologies which aim to defend against Adversarial Attacks in Black-box systems. The first one is *reactive* and the second is *proactive*. Here, reactive refers to defenses where the defender seeks to detect adversarial examples, while proactive refers to defenses where the defender seeks to make the classifier more robust. Some of the known defense strategies used to curb adversarial attacks in [9] include: 1) Preprocessing Methods 2) Regularization and Adversarial Training, 3) Distillation Methods, and 4) Classification with Rejection. Let us review a few of these methods:

Preprocessing Methods:

the authors in [17] mention this method as way filter out input determined to be adversarial. The authors argue that images have natural properties such as high correlation between adjacent pixels or low energy in high frequency. Assuming that adversarial examples and regular input do not lie in the same decision spaces can be used a pretext used to filter out malicious perturbations. According to the authors, this method is possibly not the best way to defend against adversaries since the process of filtering could potentially reduce the classifier accuracy on harmless input samples, not deemed adversarial.

Regularization and Adversarial Training:

the authors have suggested using Regularization, Adversarial Training or smoothing as a technique to robustify the classifier against adversarial attacks. The authors in [7] mention one experiment where the accuracy of the classifier fell to 17.9% with adversarial training. This type of defense cannot be relied upon when deploying security sensitive machine learning services. Regularization and Adversarial training has been shown in [9] to be effective for both Black-Box and Blind Threat Models against adversarial examples.

Distillation Methods:

in [17] Papernot proposed using a method called defensive distillation to counter adversarial examples in a Black-Box setting. This method proved useful since it limits the attacker's ability to select adversarial examples. However, there has been research which indicates that the effect of distillation can be reverted, such as work in [3]. This was made evident in the previous section where it was shown that in both threat models (Black-Box and Blind), the attacker can evade the effect of the defense method of distillation of adversarial feature blocking which the defense method depends on to thwart attacks [3]. The authors suggest that this lack of robustness is due to that notion that defensive distillation only works in the general case, but fails to protect the classifier in cases where the features are all modified at once.

2.5 Honeypots

The section focuses on the concept of Honeypots, we'll start with a basic definition of what a *honeypot* is. Then, We dissect and explain the different types of honeypots and evaluate each types' intrinsic value, as well as the different deployment types. This section also offers insight into how other security researchers have proposed using Honeypots in infrastructure security and protection.

2.5.1 Concept of Honeypots

A honeypot can be thought of as a single or group of fake systems to collect intelligence on an adversary by inducing him/her to attack it. It's meant to appear and respond like a real system, in a production environment. However, the data inside the honeypot is falsified and spurious. A honeypot has no real production value, instead its functionality is meant to record information on malicious activity. In the scenario that it should become compromised it contain no real data and therefore poses no threat on the production environment [13] [22]. As mentioned, Honeypots can be deployed with fabricated information that can be attractive to outside attackers, this characteristic can be used to re-direct attackers towards decoy systems and away from critical infrastructure [8]. See below for a typical architectural design of a honeypot system in figure 2.6.

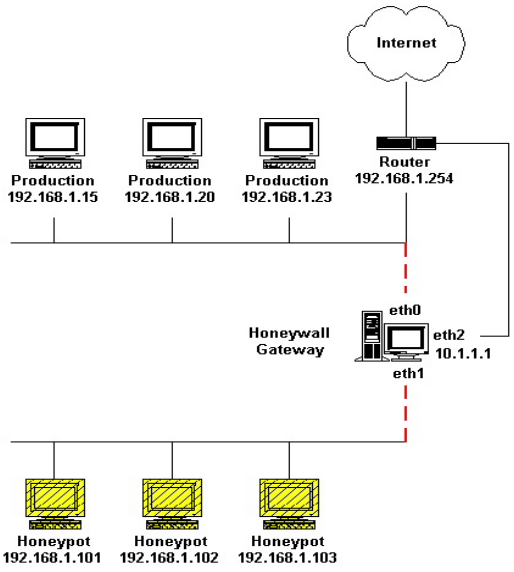

Figure 2.6: Classic Honeypot Architecture

2.5.2 Classification of Honeypots

Honeypots can be classified using several different criteria. However, for purposes of this thesis we classify them based on functionality and operation.

- **Research Honeypots** - They're Honeypots deployed with the highest level of risk associated with them, this is in order to expose the full range of attacks initiated by the adversary. They're mainly used to collect statistical data on adversarial activities inside the honeypot [13]. They're more difficult to deploy, but this does not hinder from their use by organizations to study attacks and develop security countermeasures against them. Research Honeypots help understand the trends, strategies and motives behind adversarial attacks [14].

- **Production Honeypots** - They are Honeypots known for ease of deployment and utility, and use in company production environment [14]. Closely monitored and maintained, their purpose lies in their ability to be used in an organization's security infrastructure to deflect probes and security attacks. They're attractive as an option for ease of deployment and the sheer value of information collected on the adversary.

- **Physical/Virtual Honeypots** - Physical honeypots are locally deployed honeypots, being part of the physical infrastructure. considered to be intricate and difficult to properly implement [13]. On the other hand, virtual Honeypots are simulated systems (virtualized) by the host system to forward network traffic to the virtual honeypot [14].

- **Server/Client Honeypots** - The main different between server and client honeypots is the former will wait until the adversary initiates the communication, while client Honeypots contact malicious entities and request an interaction [14]. However, traditional Honeypots are usually server-based.

- **Cloud Honeypots** - They are honeypots deployed on the cloud. This type of honeypot has many advantages, as well as restrictions. They are used by companies that at least have one part of their infrastructure on the cloud. Having the system (or part of it) in the cloud has its advantages, it makes it easy to install, update, as well as recover the honeypot in case of a corruption [13].

- **Honey-tokens** - can be thought of as a *digital* piece of information. It can manifested from a document, database entry, E-mail, or a credentials. In essence, it could be anything considered valuable enough to lure and bait the adversary. The

benefit with these tokens is that they can be used to track stolen information and level of adversarial abuse in them system [1]

2.5.3 Honeypot Deployment Modes

Honeypots can be deployed in one of three deployment modes [2], they are:

- **Deception** mode manipulates the adversary into thinking the responses are coming from the actual system itself. This system is used as a decoy and contains security weaknesses to attract attackers. According to researchers a honeypot is involved in deception activities if its responses can deceive an attacker into thinking that the response returned is from the real system.

- **Intimidation** modes is when the adversary is aware of the measures in place to protect the system. A notification may inform the the attacker that the system is protect and all activity is monitored. This countermeasure may ward or scare off any adversarial *novice*, and leave only the experienced adversaries with in-depth knowledge and competent skills to attack the system.

- **Reconnaissance** mode is used to record and capture new attacks. This information is used to implement heuristics-based rules that can be applied in intrusion detection and prevention systems. With Reconnaissance, the honeypot is used to detect both internal and external adversaries of the system.

2.5.4 Honeypot Role and Responsibilities

The true value of Honeypots lay in their ability to address the issue of security in production system environments, they mainly focus ons:

- **Interaction -** the honeypot should be responsible for interacting with the adversary. this pertains to acting as the main environment where the adversary becomes active and executes his attack strategy.

- **Deception -** the honeypot should be responsible for deceiving the adversary. This pertains to the disguising itself as a normal production environment, when in fact it's a *trap* or *sandbox* designed to exploit the adversary.

- **Data Collection -** the honeypot should be responsible for capturing and collecting data on the adversary. This information will potentially be useful for studying the attacker and his motivations.

Advantages of Honeypots

Honeypots, alone, do not enhance the security of an infrastructure. However, we can think of them as subordinate to measures already in place. However, this level of importance does not take away from some distinct advantages when compared to other security mechanisms. Here, we highlight a few [14]:

- **Valuable Data Collection** - Honeypots collect data which is not polluted with noise from production activities and which is usually of high value. This makes data sets smaller and data analysis less complex.
- **Flexibility** - Honeypots are a very flexible concept to comprehend, as can be seen by the wide array of honeypot software available in the market. The indicates that a well-adjusted honeypot tool can be modified and used for different tasks, which further reduces architecture redundancy.
- **Independent from Workload** - Honeypots do not need to process traffic directed or which originates from them. This means they are independent from the workload which the production system experiences.
- **Zero-Day-Exploit Detection** - Honeypots capture any and every activity occurring within them, this could give indication to unseen adversarial strategies, trends and zero-day-exploits that can be identified from the session data collected.
- **Lower False Positives and Negatives** - any activity that occurs inside the server-honeypot is a considered to be out-of-place and therefore an anomaly, which is by definition an attack. Honeypots verify attacks by detecting system state changes and activities that occur within the honeypot container. This helps to reduce false positives and negatives.

Disadvantages of Honeypots

Ultimately, no one security system or tool that exists is faultless. Honeypots suffers from some disadvantages, some of them are [14]:

- **Limited Field of View** - a Honeypots is only useful if an adversary attacks them, and worthless if no one does. if the honeypot is evaded by the adversary, and attacks the production system or target environment directly, it will not be detected.

- **Being Fingerprinted** - here, fingerprinting signifies the ability of the attacker to identify the presence of a honeypot. If the honeypot behaves differently than a real system, the attacker might identify and consequently detect it. If their presence is detected, the attacker can simply ignore the honeypot and attack the targeted system instead.

- **Risk to the Environment** - Honeypots might introduce a vulnerability to the production infrastructure environment, if exploited and compromised. And naturally, as the level of interaction (freedom) that the adversary has within the environment increases, so does the level of potential misuse and risk. The honeypot can be monitored, and the risk mitigated, but not completely eliminated.

2.5.5 Honeypots Level of Interaction

A honeypot is considered to be an fake system, with no real value. It is built and designed to emulate the same tasks that a real production system can accomplish. However, these tasks are of no significance, hence compromising the honeypot poses no threat on the production environment .Honeypot system functionality can be categorized according to the level interaction the adversary has with the honeypot system environment [13]:

- **Low-interaction Honeypot(LIHP) -** these type of system emulate only simple services like *Secure Shell* (SSH), *Hypertext Transfer Protocol*(HTTP) or *File Transfer Protocol*(FTP). These systems are easily discoverable by attackers and provide the lowest possible level of security. However, they have a promising advantage, they are easy to install, configure and monitor. They should not be used in production environments, but for education and demonstration purposes. Some examples of such systems include *Honeyperl*, *Honeypoint*, and *mysqlpot*. See a typical architectural design of a low-interaction honeypot in Figure 2.7 below.

- **Medium-interaction Honeypots(MIHP) -** this type of system is a hybrid, which lays in the middle ground between low/high interaction honeypots. This means that the honeypot is still an instance that runs within the operating system. However it blends in so seamlessly into the environment that it becomes difficult to detect by attackers lurking within the network. Some examples of such systems are *Kippo* and *Honeypy*.

- **High interaction Honeypot(HIHP) -** the main characteristic regarding High-Interaction Honeypots is that they're using a real live operating system. It uses more hardware resources and poses a major level risk on the rest of the production

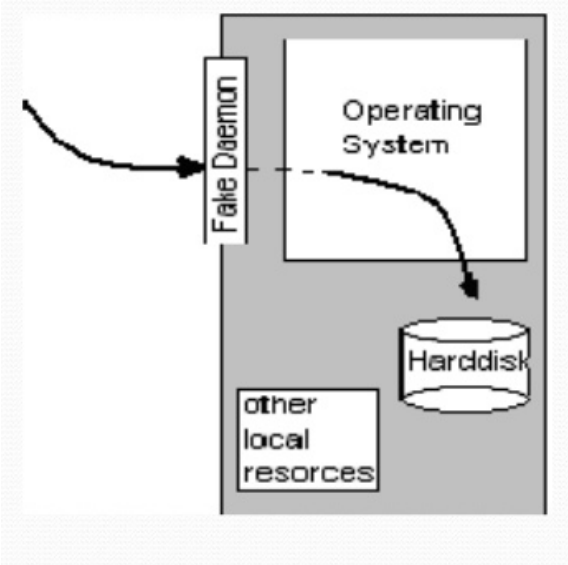

Figure 2.7: Low Interaction Honeypot

environment and infrastructure, when deployed. In order to minimize risk and prevent exploitation by an adversary, it is constantly under monitoring. Some examples of such systems are *Pwnypot* and *Capture-HPC*. See a typical architectural design of a high-interaction honeypot in Figure 2.8 below.

2.5.6 Uses of Honeypots

As mentioned above, honeypots have a wide array of enterprise applications and uses. Currently, honeypot technology has been utilized in detecting *Internet of Things* (IoT) cyberattack behavior, by analyzing incoming network traffic traversing through IoT nodes, and gathering attack intelligence [5]. In robotics, a honeypot was built to investigate remote network attacks on robotic systems [10]. Evidently, There is an increasing need to install *red herring* system in place to thwart adversarial attacks before they occur and cause damage to production systems.

One of the most popular type of honeypots technologies witnessing an increase in its popularity is High-Interaction-Honeypots (HIHP). This type of honeypot is preferred since it provides a real-live system for the attacker to be active in. This property is valuable, since it captures the full spectrum of attacks launched by adversaries within the system. It us allows to learn as much as possible about the attacker, the strategy involved and tools used. Gaining this knowledge allows security experts to get insight into what future attacks might look like, and better understand the current ones.

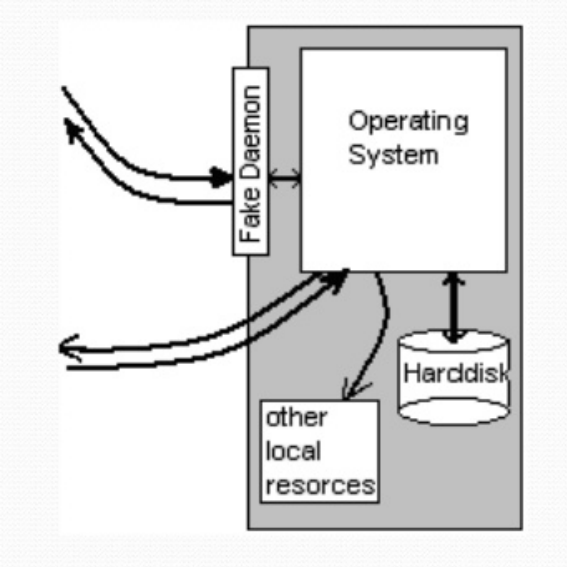

Figure 2.8: High Interaction Honeypot

In the next chapter, we will explore and discuss the work done by other researchers in the areas of Black-Box systems defense and deception as a method of defense.

2.6 Honeypot in our Solution

We formulate the problem of devising an supplementary method of defense against Adversarial Examples. This secondary level of defense will shield the Black-Box System, using honeypots as the primary tool of deception in building the system.
This decentralized and distributed framework must consist of N of high-interaction honeypots. Each of these N honeypots is embedded with a decoy target model $\check{\mathrm{T}}$ designed to lure and prevent an adversary with adversarial input $O'\vec{x}$ from succeeding in causing a mislabeling attack on the target model T. Essentially, the framework must perform the following tasks below.
Firstly, prevent the adversary from mimicking the neural network behavior and replicating the decision space of the model. This will be done by essentially blocking adversarial transferability, prevent the building of the substitute model F from occurring, and the transfer of samples from the substitute model F to the target model T. This makes it difficult to find a perturbation that satisfied $O\{\vec{x} + \delta\vec{x}\} = O\{\vec{x}\}$
Secondly, the framework must lure the adversary away from the target model T, using

deception techniques. These methods consist of using: 1) deployment of uniquely generated digital breadcrumbs (HoneyTokens), 2) setting weak TCP/IP ports open to attract the adversary scanning the network, and 3) set up decoy target models $\check{\mathrm{T}}$, deployed inside the honeypots for the attacker to interact with, instead of the actual target model T.

Finally, create an infeasible amount of computational work for the attacker, with no useful outcome or benefit. This can be accomplished by presenting the attacker with the non-convex, non-linear, and hard optimization problem, which is generating adversarial samples to transfer to the target model, which in this case is a decoy; a decoy of the same optimization problem below:

$$\vec{x^*} = \vec{x} + \arg\min\{\vec{z} : \tilde{O}(\vec{x} + \vec{z}) \neq \tilde{O}(\vec{x})\} = \vec{x} + \delta_{\vec{x}}$$

Figure 2.9: Adversarial Example Optimization Problem

This strenuous task is complicated further for the attacker because in order to generating the synthetic samples, the attacker must approximate the unknown target model architecture and structure to train the substitute model F, which is challenging. Evasion is further complicated as the number of deployed honeypots N in the framework increases. Therefore, building this system consists of solving three problems in one, preventing of adversarial transferability, deceiving the attacker and creating immense computational work for adversary targeting the system to waste the adversary time and resources. All the later, while keeping the actual target model T out of reach.

Chapter 3

Related Work

The purpose of this chapter is to summarize and evaluate the authored work of other researchers on techniques and frameworks designed to defend Black-Box systems from adversarial example attacks. This chapter also touches on how deception as a defense technique in used to protect security systems, specifically in the use of fake digital entities to attract and deceive adversaries.

The works below focus directly on the concept of defending against adversarial examples aimed at misleading the classifier. Defense is accomplished by data preprocessing during the training phase of the neural net model preparation. That typically means influencing the effect that data will have on the underlying DNN model, and filtering out malicious perturbations inserted by an adversary that may corrupt it. Other works in this section focus on the role of Cyber Security defense through method of deception, specifically on the role of decoys and fake entities to deceive the attacker. Our challenge here is construct a secondary level of protection and defense, designed not to replace existing defense technique, but supplement and reinforce the aforementioned defense frameworks below, through means of adversarial deception.

An alternative, but conceptually close to the data preprocessing technique is, the use of distillation as a defense against adversarial perturbations [17], and reduce the impact impurities make on the neural net features. Here, the authors use this simple technique to reduce the dimensionality of DNN classifier model, while maintaining accuracy. However, the work in [3] suggest that a model protected using defense distillation is no more secure than a model without any defense. While it is able to defend against some cases attacks, it cannot protect the model neural net against all types of attack that occur at once.

The following papers and works deal directly with defenses against adversarial examples and other works associated with defense through deception using HoneyTokens:

i. *Efficient Defenses Against Adversarial Attacks* - this paper [25] focuses on addressing the lack of efficient defenses against adversarial attacks that undermine and the fool Deep Neural Networks (DNNs). The need to tackle this hurdle has been amplified by the fact that there isn't a unified understanding of how or what makes these DNN models so vulnerable to attacks by adversaries. The authors propose an effective solution which focuses on re-reinforcing the existent DNN model and making it robust to adversarial attacks, attempting to fool it. The proposed solution focuses on utilizing two strategies to strengthen the model, which can be used separately or together. The first strategy is using bounded ReLU activation functions in the DNN architecture to stabilize the model, and second is based on augmented Gaussian data for training. Defenses based on data augmentation improves generalization since it considers both the true input and its perturbed version. The latter enables a broader range of search in the input, then say, adversarial training, which is limited and only uses part of the input, falling short. The result of applying both strategies results in a much smoother and stable model, without losing on the model's performance or accuracy.

ii. *Blocking Transferability of Adversarial Examples in Black-Box Learning Systems* - this paper [9] is the most relevant academic paper, with regards to motivation and stimulus for purposes of developing our proposed auxiliary defense technique with honeypots. The authors in [9] propose an training approach aimed at robustifying Black-Box learning systems against adversarial perturbations, by blocking transferability, which covertly transfers perturbed input between classifiers. The proposed method of training called *NULL labeling*, works by evaluating input and lowering confidence on the output label if suspected to be perturbed. The training method smoothly filters out and discards the invalid input, instead of allowing it to be classified into intended target label. The ingenuity of this approach lies in how it is able to distinguish between clean and malicious input. This method proves its capability in blocking adversarial transferability and resisting the invalid input that exploit it. The latter is achieved by mapping malicious input to a NULL label and allowing clean test data to be classified into its original label, all while maintaining prediction accuracy.

iii. *Towards Robust Deep Neural Networks with BANG* - this paper [20] is another training approach for combating adversarial examples and robustifying the learning model. The authors propose this technique in response to the unknown reason for the existence of adversarial examples, not to mention their abnormal and mys-

terious nature. The authors argue that regular adversarial training still make the model vulnerable and exposed to adversarial examples. For this very purpose, the authors present a data training approach, known as *Batch Adjusted Network Gradients* or *BANG*. This method works by attempting to balance the causality that each input element has on the node weight updates. This efficient method achieves enhanced stability in the model by forming *smoother* areas in the classification region and becomes robust to input perturbations that work on exploiting and violating the integrity of the model. This method is designed to avoid instability brought about by the adversarial examples. This training method achieves good results and has low computational cost, while maintaining classification accuracy.

iv. *HoneyCirculator: distributing credential HoneyToken for introspection of web-based attack cycle* - in this paper [1] the authors suggest a framework that actively and purposely leaks digital entities in the network to deceive the adversaries and lures them to a honeypot, which is actively under monitoring, tracking token access, and watching out for new adversarial trends. In a period of 1 year, the system was compromised by multiple adversaries, without being identified as a ruse. This framework allows the constant surveillance of adversaries, without being recognized as a decoy. The authors argue that this method is successful, as long as the attacker does not change his attack strategy. However, a main concern for the authors is designing convincing enough fake data to deceive, attract, and fool an adversary. The authors argue that the defender should design fake entities that are *attractive* enough to bait the attacker, while not revealing important or compromising information to the attacker, and learn as much as possible about the attacker. As the threat of adversarial attacks increases, so will the need for novelty in the approach to combat it.

v. *A Survey on Fake Entities as a Method to Detect and Monitor Malicious Activity* - this survey paper [18] serves as an examination of the concept of *fake entities* and digital tokens, which my thesis defense relies partially focuses on. Fake entities, although primitive, are an attractive asset in any security system. Fake entities could be files, interfaces, memory, database entries, meta-data, etc. For the authors, this inexpensive, lightweight and easy-to deploy *pawns* are as valuable as any of the other security mechanisms in the field. Simply put, they're digital objects, embedded with fake divulged information, intended to be accessed by the attacker. For the authors, operating system based fake entities are the most attractive and fitting to become a decoy, due to the variety of ways the operating system interface can

be configured. Once in possession by the attacker, the defender is notified and can begin monitoring the attacker's activity. The authors implemented a framework that actively leaks credentials and leads adversaries to a controlled and monitored honeypot. The authors have yet to build a proof of concept.

There is also extensive work done on utilizing adversarial transferability in other forms of adversarial attacks, deep learning vulnerabilities in DNNs, and black-box attacks in machine Learning. Other interesting works that served as a motivation for this thesis include, utilizing honeypots in defense techniques, such as design and implementation of a honey-trap [6], deception in distributed system environments [21], and using containers in deceptive honeypots [11].

As mentioned, our approach, using Honeypots, does not seek to replace any of the existing methods to combat adversarial examples in black-box setting. However, it can effectively be used as an auxiliary method of protection that *laminates* existing defense methods. below, we briefly outline the limitations in our method, as well as how it differs from other defense methods and techniques in the following ways:

3.1 Novelty in our approach

3.2 Limitations

3.3 Assumptions

Chapter 4

Proposed Defense Approach

4.1 Motivation

4.2 Approach

4.3 Attacker

4.3.1 Attack Setting

4.3.2 Attack Goal

4.3.3 Attacker Knowledge

4.3.4 Attacker Capabilities

4.4 Adversarial Honeypot Network Overview

4.5 Individual Honeypot Topology

4.6 Threat Model

4.7 Target and Decoy Model

4.8 Target Model

4.8.1 Purpose

4.8.2 Architecture and Topology

4.8.3 Training, Testing and Validation

4.9 Attracting The Adversary

4.9.1 Adversarial Tokens

4.9.2 Weak TCP/IP ports

4.9.3 Decoy Target Model

4.10 Detecting Malicious Behavior

4.11 Monitoring the Adversary

4.12 Launching The Attack

4.13 Defending Against Attack

4.14 Deployment

4.15 Scalability

4.16 Security

4.17 Mathematical models of virus dynamics

The basic model of virus dynamics can be written as follows [**?**, **?**, **?**]:

$$\begin{aligned} \frac{\mathrm{d}T}{\mathrm{d}t} &= -\beta TV \\ \frac{\mathrm{d}I}{\mathrm{d}t} &= \beta TV - \delta I \\ \frac{\mathrm{d}V}{\mathrm{d}t} &= pI - cV \ . \end{aligned}$$

where T is something something.

4.18 Shape of the viral titer curve

The virus spread and kills everything. This is well illustrated in Figure 4.1 where the kinetics of the infection are shown for three different viral production rates of the sec-

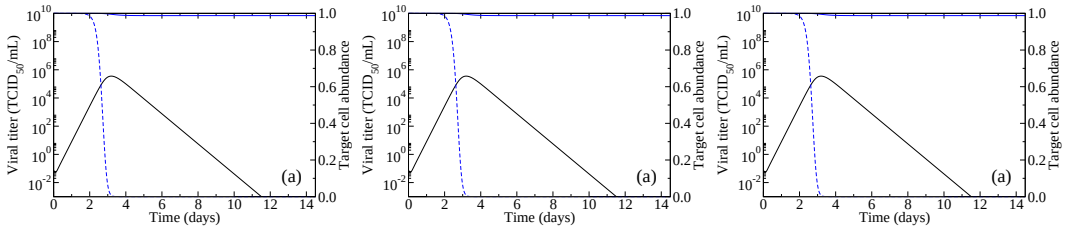

Figure 4.1: **An example illustrating how to include figure files.** I can not only display figures, but I can resize them so they look just right to appear side-by-side. This is how you produce multiple-panel images as a single figure. In addition, if your figure file is generated from some sort of model, you can refer to the table where the data or model parameters are listed. For example, you could say: All parameters are as in Table **??**.

ondary cell population for the case where these cells are 1,000-fold harder to infect than cells of the default type. When secondary cells produce only 10-fold more virus than cells of the default type, the infection is mostly limited to the default cell population as the amount of virus produced is not sufficient for the infection to spread to the secondary cell population. Increasing the production rate to 100-fold more than cells of the default type results in a sufficient amount of virus being produced to sustain a slow growing infection within the secondary cell population, leading to long-lasting, high-levels of viral titer. Finally, increasing the viral production rate to 1,000-fold more than the default cell type allows the infection to successfully infect and decimate both cell populations rapidly. From these results, we see that there appears to be a relationship between the secondary cells' susceptibility to infection and their viral production rate which leads to a severe and sustained infection.

Note that all parameters use to produce our simulations can be found in Table **??**.

Chapter 5

Evaluation and Discussion

Chapter 6

Implementation and Discussion

6.1 Background

6.2 Architecture

6.3 Features

6.4 Functionality

6.5 Usage

6.6 Deployment

6.7 Integration

6.8 Benefits

6.9 Future Work

Appendix A

Summary and Contributions

A.1 Summary

A.2 Discussion

A.3 Contributions

A.4 Future Work

A.5 Conclusion

Appendix B

Appendix A

Appendix C

Appendix B

blah blah

C.0.1 An appendix subsection if required

blah blah blah.

Appendix D

The second appendix chapter

D.1 A section in my second appendix chapter

blah blah.

D.1.1 Just making sure it all works

blah blah blah.

Bibliography

[1] M. Akiyama, T. Yagi, T. Hariu, and Y. Kadobayashi. HoneyCirculator: distributing credential honeytoken for introspection of web-based attack cycle. *Int. J. Inf. Secur.*, pages 1–17, Jan. 2017.

[2] R. M. Campbell, K. Padayachee, and T. Masombuka. A survey of honeypot research: Trends and opportunities. In *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 208–212, Dec. 2015.

[3] N. Carlini and D. Wagner. Defensive Distillation is Not Robust to Adversarial Examples. *arXiv:1607.04311 [cs]*, July 2016. arXiv: 1607.04311.

[4] N. Carlini and D. Wagner. Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods. *arXiv:1705.07263 [cs]*, May 2017. arXiv: 1705.07263.

[5] S. Dowling, M. Schukat, and H. Melvin. A ZigBee honeypot to assess IoT cyberattack behaviour. In *2017 28th Irish Signals and Systems Conference (ISSC)*, pages 1–6, June 2017.

[6] A. A. Egupov, S. V. Zareshin, I. M. Yadikin, and D. S. Silnov. Development and implementation of a Honeypot-trap. In *2017 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, pages 382–385, Feb. 2017.

[7] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and Harnessing Adversarial Examples. *arXiv:1412.6572 [cs, stat]*, Dec. 2014. arXiv: 1412.6572.

[8] J. D. Guarnizo, A. Tambe, S. S. Bhunia, M. Ochoa, N. O. Tippenhauer, A. Shabtai, and Y. Elovici. SIPHON: Towards Scalable High-Interaction Physical Honeypots. In *Proceedings of the 3rd ACM Workshop on Cyber-Physical System Security*, CPSS '17, pages 57–68, New York, NY, USA, 2017. ACM.

[9] H. Hosseini, Y. Chen, S. Kannan, B. Zhang, and R. Poovendran. Blocking Transferability of Adversarial Examples in Black-Box Learning Systems. *arXiv:1703.04318 [cs]*, Mar. 2017. arXiv: 1703.04318.

[10] C. Irvene, D. Formby, S. Litchfield, and R. Beyah. HoneyBot: A Honeypot for Robotic Systems. *Proceedings of the IEEE*, PP(99):1–10, 2017.

[11] A. Kedrowitsch, D. D. Yao, G. Wang, and K. Cameron. A First Look: Using Linux Containers for Deceptive Honeypots. In *Proceedings of the 2017 Workshop on Automated Decision Making for Active Cyber Defense*, SafeConfig '17, pages 15–22, New York, NY, USA, 2017. ACM.

[12] A. Kurakin, I. Goodfellow, and S. Bengio. Adversarial examples in the physical world. *arXiv:1607.02533 [cs, stat]*, July 2016. arXiv: 1607.02533.

[13] M. A. Lihet and V. Dadarlat. How to build a honeypot System in the cloud. In *2015 14th RoEduNet International Conference - Networking in Education and Research (RoEduNet NER)*, pages 190–194, Sept. 2015.

[14] M. Nawrocki, M. Whlisch, T. C. Schmidt, C. Keil, and J. Schnfelder. A Survey on Honeypot Software and Data Analysis. *arXiv:1608.06249 [cs]*, Aug. 2016. arXiv: 1608.06249.

[15] N. Papernot, P. McDaniel, and I. Goodfellow. Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples. *arXiv:1605.07277 [cs]*, May 2016. arXiv: 1605.07277.

[16] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami. Practical Black-Box Attacks against Machine Learning. *arXiv:1602.02697 [cs]*, Feb. 2016. arXiv: 1602.02697.

[17] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami. Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 582–597, May 2016.

[18] S. Rauti and V. Leppnen. A Survey on Fake Entities as a Method to Detect and Monitor Malicious Activity. In *2017 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, pages 386–390, Mar. 2017.

[19] M. Ribeiro, K. Grolinger, and M. A. M. Capretz. MLaaS: Machine Learning as a Service. In *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, pages 896–902, Dec. 2015.

[20] A. Rozsa, M. Gunther, and T. E. Boult. Towards Robust Deep Neural Networks with BANG. *arXiv:1612.00138 [cs]*, Nov. 2016. arXiv: 1612.00138.

[21] N. Soule, P. Pal, S. Clark, B. Krisler, and A. Macera. Enabling defensive deception in distributed system environments. In *2016 Resilience Week (RWS)*, pages 73–76, Aug. 2016.

[22] X. Suo, X. Han, and Y. Gao. Research on the application of honeypot technology in intrusion detection system. In *2014 IEEE Workshop on Advanced Research and Technology in Industry Applications (WARTIA)*, pages 1030–1032, Sept. 2014.

[23] F. Tramr, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel. The Space of Transferable Adversarial Examples. *arXiv:1704.03453 [cs, stat]*, Apr. 2017. arXiv: 1704.03453.

[24] J. D. Tygar. Adversarial Machine Learning. *IEEE Internet Computing*, 15(5):4–6, Sept. 2011.

[25] V. Zantedeschi, M.-I. Nicolae, and A. Rawat. Efficient Defenses Against Adversarial Attacks. *arXiv:1707.06728 [cs]*, July 2017. arXiv: 1707.06728.