# Using Honeypots in a Decentralized Framework to Defend Against Adversarial Machine-Learning Attacks

Fadi Younis and Ali Miri

Department of Computer Science
Ryerson University, Toronto, Ontario, Canada
`fyounis, ali.miri@ryerson.ca`

**Abstract.** The market demand for online machine-learning services is increasing, and so have the threats against them. Adversarial inputs represent a new threat to *Machine-Learning-as-a-Services* (MLaaSs). Meticulously crafted malicious inputs can be used to mislead and confuse the learning model, even in cases where the adversary only has limited access to input and output labels. As a result, there has been an increased interest in defence techniques to combat these types of attacks. In this paper, we propose a network of *High-Interaction Honeypots* (HIHP) as a decentralized defence framework that prevents an adversary from corrupting the learning model. We accomplish our aim by 1) preventing the attacker from correctly learning the labels and approximating the architecture of the black-box system; 2) luring the attacker away, towards a decoy model, using *Adversarial HoneyTokens*; and finally 3) creating infeasible computational work for the adversary.

**Keywords:** adversarial machine learning; deception-as-a-defense; exploratory attacks; evasion attacks; high-interaction honeypots; honeytokens

## 1 Introduction

Recent years has seen an exponential growth in the utilization of machine-learning tools in critical applications, services and domains. This has led to many service providers now offering their machine-learning products in the form of online cloud services, known as *Machine-Learning-as-a-Service* (MLaaS) [20]. These services allow user to simply use these tools, without the efforts needed to train, test, and fine-tune the underlying machine-learning models. While some more experienced users may still prefer to control how their models are constructed and deployed, to protect their models MLaaS providers typically hide the complex internal mechanisms from most of their users, and simply package the services in non-transparent and obfuscated ways. That is they provide their services in the form of a *black-box* [12] [18]. This opaque system container accepts some input and produces an output, but in this system the internal details of the prediction model are hidden from the user. Although hiding the internal mechanisms of models used can provide some protection against insider and outsider attacks, these types of deployments remain susceptible to attacks. For example, an attacker can try to mislead and confuse the MLaaS prediction model, using specifically crafted examples, known as *adversarial examples* [12], leading to a violation of the model integrity [18]. Most proposed defences against these types of attacks aim to strengthen the underlying model by training it against possible expected adversarial malicious inputs. These approaches - such as *Regularization* and *Adversarial Training* [26] - may have limited success, as they do not generalize against newer and more complex adversarial inputs. In this paper, we propose a new defence framework, that can provide an additional layer of protection for MLaaS services. As an example, we show how our framework can be used to defend against malicious attacks on *Artificial Neural Network* (ANN) classifiers. It has been shown that adversarial attacks on these type of classifiers can go undetected [15]. Maliciously crafted adversarial examples can be used to exploit *blind spots* in the classifier boundary space. Exploiting these blind sports can be used to mislead and confuse the learning mechanism in the classifier, post model training, for purposes of violating model integrity.

Our challenge here lies in constructing an adversarial defence technique capable of dealing with different, and possibly adaptive types of attacks. Part of our defence framework utilizes *adversarial HoneyTokens*, fictional digital *breadcrumbs* designed to lure the attacker. They are made conspicuously detectable, to be discovered by the adversary. It is possible to generate a unique token for each item (or sequence) to deceive the attacker and track his abuse. However each token must be strategically designed, generated and deliberately embedded into the system , to misinform and fool the adversary. A major component of our defence framework is focused on designing a decentralized network of *High-Interaction Honeypots* (HIHP), as an open target for adversaries, acting as a type of *perimeter* defence. This decentralized network of honeypot nodes act as self-contained *sandboxes*, to contain the decoy neural network, collect valuable data, and potentially gain insight into adversarial attacks. We believe this can also confound and deter adversaries from attacking the target model to begin with. Other adversarial defenses can also benefit by utilizing this framework as an additive layer of security to their techniques to protect production servers where learning models reside. Unlike other defense models proposed in literature, we have designed our defense framework to deceive the adversary in three consecutive steps, occurring in strategic order. The information collected from the attacker's interaction with the decoy model could then potentially be used to learn from the attacker, re-train and fortify the deep learning model in future training iterations, but for now this falls out outside of our scope.

The contributions of this paper are the following:

– We propose an adversarial defence approach that will act as a secondary-level of protecting to *cloak* and reinforce existing adversarial defense mechanisms. This approach aims to: 1) prevent an attacker from correctly learning the classifier labels and approximating the correct architecture of the black-box system; 2) lure attackers away from the target model towards a decoy model, and re-channel adversarial transferability; 3) create unfeasible computational

work for the adversary, with no functional use or benefit, other than to waste his resources and distract him while learning his techniques.
- We provide an architecture and extended implementation of the *Adversarial HoneyTokens*, their designs, features, usage, deployment benefits, and evaluations.

This paper focuses on using honeypots in defending against adversarial attacks against machine learning techniques, and in particular deep learning. For completeness, we have included some background and relevant concepts such as *adversarial examples*, *adversarial transferability* and *black-box learning systems* in the appendix. The rest of this paper is organized as follows. In *Section 2*, we present the role of honeypots in our approach, threat models, attack environments and settings. In *Section 3* , we present our 3-tier defence approach. In *Section 4* we will discuss the related work, followed by conclusions and future work in *Section 5*.

## 2 Threat Model and Settings

### 2.1 Problem Definition

The main goal of this paper to build a decentralized adversarial defense framework against adversarial examples. This level of defense will shield the black-box learning system, using honeypots as one of the primary components of deception in building the framework. This decentralized framework must consist of $H$ high-interaction honeypots. Each of these honeypots is embedded with a decoy target model $T_{decoy}$, designed to lure and prevent an adversary with adversarial input $\boldsymbol{x}$ from succeeding in causing a mislabeling attack $f(x) = y_{true}$ on the target model $T_{target}$. Essentially, the framework must perform the following tasks below.

- *task 1* - prevent the adversary from mimicking the neural network behavior in the learning function *f()* and replicating the decision space of the model. This will be done by blocking adversarial transferability, prevent the building of the correct substitute training model $F(S_p)$ from occurring and the transfer of samples from the substitute model $F$ to the target model $T_{target}$. This makes it difficult to find a perturbation that satisfies $O\{\boldsymbol{x} + \delta\boldsymbol{x}\} = O\{\boldsymbol{x}\}$, since the target model duplicated is fake.
- *task 2* - 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) $TK_n$, 2) making the network of honeypot nodes easily accessible 3) set up decoy target models $T_{decoy}$, deployed inside the honeypots for the attacker to interact with, instead of the actual target model $T_{target}$.
- *task 3* - create an in-feasible amount of computational work for the attacker, with no useful outcome or benefit. This can be accomplished by presenting the attacker with a *non-convex*, *non-linear* and hard optimization problem,

which is generating adversarial samples to transfer to the remote target model $T_{target}$, which in this case is a decoy; a decoy of the same optimization problem we saw in the earlier sections:

$$\boldsymbol{x}^* = \boldsymbol{x} + argmin\{\boldsymbol{z} : \hat{O}(\boldsymbol{x}+\boldsymbol{z}) \neq \hat{O}(\boldsymbol{x})\} = \boldsymbol{x} + \delta_x$$

This strenuous task is complicated further for the attacker because in order to generate the synthetic samples, the attacker must approximate the unknown target model architecture and structure $F$ to train the substitute model $F(S_p)$, which is challenging. Evasion is further complicated as the number of deployed honeypots in the framework increases. Therefore, building this system consists of solving three problems in one, preventing of adversarial transferability[1], deceiving the attacker and creating immense computational work for adversary targeting the system to waste computational time and resources; all the later, while keeping the actual target model $T_{target}$ out-of-reach.

Hence, the adversarial examples generated need to have such an effect on the classifier, that it explicitly lowers the confidence on the target label. Misclassification attacks, to us, were less attractive since they do not make for interesting adversaries, not to mention the fact that these type of attacks appear random in nature, focusing on an arbitrary set of data samples. With no fringe inconsistencies to dispute, it becomes difficult to discern failures brought about by non-malicious factors effecting the classifier. Building on the latter, misclassification attacks make it all the more difficult to design defenses and robust frameworks to thwart adversaries when the attack itself seems arbitrary in nature.

## 2.2 Threat Model

**Attack Specificity -** generally, for an adversary to succeed in his attack, and whether the attacker has his sight set on violating the *availability* or *integrity* of the model, adversarial transferability needs to be successful. For purposes of our paper, we have decided to design our adversarial attack to be a *targeted exploratory* one in nature [9]. A targeted attack is when the adversary has a specific set of data samples in mind, and is discriminatory in his attack. This means the adversary wants to force the DNN to output a specific target label $y_{target}$, $f(x) \longrightarrow y_{target}$, instead of the correct label $y_{true}$, $f(x) \nrightarrow y_{true}$. See Figure 1 for an illustration of a adversarial targeted attack, violating model integrity.

**Exploited Vulnerability -** the cogent properties of adversarial examples $\boldsymbol{x}^*$ make them a prime candidate for adversarial attacks on deep learning systems. It should be anticipated that an ambitious and equally resourceful adversary will conspire to use these perturbations for malicious purposes. Generally, deep neural nets (DNN) work by extracting and learning the key multi-dimensional

[1] Even with the little knowledge possessed by a potential adversary, a targeted attack in a black-box setting is still in fact probable

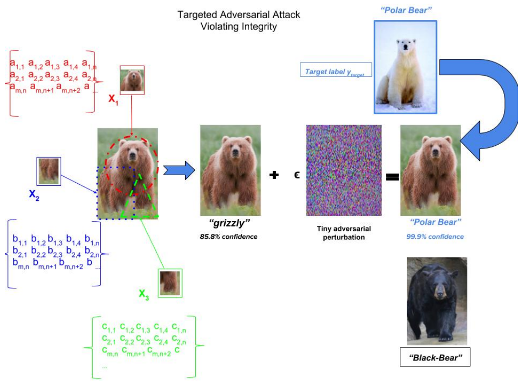

**Fig. 1.** Input $\boldsymbol{x}$ (left), modification $\delta + \boldsymbol{x}$ controlled by $\varepsilon$ (middle) which controls the magnitude of modification in the image, generating the adversarial evading sample $\boldsymbol{x}^*$(right). As you can see, both bus images look astoundingly similar.

discriminate features $X_{m,n} = \{x_{n,1}, x_{n,2}, x_{n,3}, ..., x_{n,m}\}$ embedded within the input sample $x$ pixels, to correctly classify it with the correct output label $y_{true}$. However, with adversarial examples entities, the acuity of a DNNs classification ability becomes slightly manipulable, and the adversary is aware of this weaknesses.
In our paper, the designed adversary's attack depends on the successful exploitation of a fundamental vulnerability found in most, if not universally all DNN learning systems. This vulnerability is acquired during faulty model training. This weakness is embodied by a lack of non-linearity in poorly trained DNN models, that these visually indistinguishable adversarial examples, born in a high-dimensional space, epitomize. Other factors may also be responsible, such as poor model regularization. This inability to cope with non-linearity makes the DNN classifier insensitive to certain blind-spots in the high-dimensional classification region. Knowing the latter, an adversary can generate impressions of the input samples with slight perturbations. These examples can then be transferred between adjacent models, due to the cross-model-generalization property which allow the transfer of adversarial examples between the original and target model the adversary desires to exploit. The above vulnerability is manifested after the examples are synthesized and injected during the testing phase.

**Attacker Capabilities -** each honeypot node in the decentralized defense framework contains a decoy target model $T_{decoy}$, presented to the adversary as the legitimate target model. Here, an *Oracle O* represents the means for the adversary to observe the current state of the DNN classifier learning by observing how a target model $T_{target}$ handles the testing sample set $(x^{'}, y^{'})$. In our attack environment, querying the *Oracle O* with queries $q = \{q_1, q_2, q_3, ..., q_n\}$ is the exclusive and only capability an adversary possesses for learning about the target model and collecting his synthetic dataset $S_p$ to build and gradually train his DNN substitute model $F$. The adversary can create a small synthetic set of adversarial training samples from the initial set $S_0$ with output label $y^{'}$ for any input $x^{'}$ by sending $q_n > 1$ queries to the *Oracle O*. The output label $y^{'}$ recurred is the result of assigning the highest probability assigned a label $y^{'}$ which maps back to a given $x^{'}$ is the only capability that the attacker has for learning about presumed target model $T_{target}$ through its *Oracle O*. The attacker has virtually no information about the DNN internal details. The adversary is restrained by the same restrictions a regular user querying the *Oracle O* has. The latter is something an adversary should adhere to make his querying attempts seem harmless, while engaging the decoy model within the adversarial honeypot. Finally, we anticipate that the adversary will not restrict himself to querying one model and will likely connect to multiple nodes and DNN model classifiers from the same connection for purposes of synthetic data collection in parallel. This should trigger an alarm within our framework, indicating multiple access and that something abnormal is occurring.

## 2.3 Attack Setting

Our envisioned profile for the adversary targeting our black-box learning system does not possess any internal knowledge regarding the core functional components of the target model $T_{target}$ DNN. This restriction entails no access to model's *DNN architecture, model hyper-parameters, learning rate, etc.* We have already established that an adversary can prepare for an attack by simply monitoring target model $T_{target}$ through its *Oracle O* and use the labels to replicate and train an approximated architecture $F$.
The ad-hoc approach at the adversary's disposal is that he can learn the corresponding labels by observing how the target model $T_{target}$ classifies them during the testing phase. The adversary can then build his own substitute training model $F$ and use this substitute model $F$ in conjunction with synthetic labels $S_p$ to generate adversarial examples propped against the substitute classifier, which the attacker has access to. Even if the substitute model $S$ and target model $T_{target}$ are different in architecture, the adversarial examples $x^{*}$ generated for one can still tarnish the other if transferred using *adversarial transferability*. Since the adversarial examples between both models are only separated by added tiny noise $\varepsilon$, the examples look similar in appearance. The latter is true even if both models, original $T_{target}$ and substitute model $F$, differ in architecture and training data. As long as both models have the same purpose and model type. Although the *Adversarial transferability* phenomena is discouraging, but alone it is advantageous for the adversarial attackers to launch targeted attacks, with little or no constraint on their attack blueprint. Adversarial transferability eventually becomes a serious concern because attacks will grow in sophistication and potency over time. It is challenging to design a model that can generalize against more advanced attacks, if not all. Also, it is difficult to dismantle and reverse-engineer how these attacks propagate and cause harm, since no tools exist to expedite the process to learn from the attack in time to re-train the network.

## 3 Deception-As-A-Defense Approach

The proposed *Adversarial Honeynet* framework is considered as an added layer of protection to *blanket* a deployed deep learning system, in order to combat imperceptible adversarial examples, within a black-box attack setting. There are several advantages and benefits that this framework can bring in the protection of existing learning systems. A single adversarial honeypot node in this decentralized framework may offer the following benefits: 1) *adversarial re-learning*; conceptually, it is a pragmatic method of collecting intelligence on the adversary, such as attack patterns, propagation, frequency and evolution. The latter results can be used to learn and reverse-engineer adversarial attacks; 2) an *anomalous classifier* used to identify whether the attackers actions are malicious or benign, this will help to determine whether or not to record the attackers session information based on behavior patters against a *white-list*; 3) a *decoy target model*, used as a placeholder for the adversary to engage and interact in case his intention are indeed malicious in nature. The attacker's interaction with model is represented by the *Oracle* $\hat{O}$, that an adversary observes and queries, re-channeling his efforts; 4) an *Adversarial Database*, used to collect and securely store attack session data on the adversary's actions and maneuvers, used later to research and understand the adversary in *adversarial re-learning*.

### 3.1 Adversarial Honeynet

All honeypot nodes are deployed with identical decoy models $T_{decoy}$ that resemble the original target DNN model $T_{target}$. Also, all services and applications on the high-interaction honeypot are real and not simulated, prompting the attacker to assume the model is indeed real, published or leaked by mistake. Neighboring adversarial honeypots are called *HoneyPeers*, these nodes are always active and have a weak non-privileged TCP/IP port open that is known to attract adversaries, supported with adversarial honeytokens. The docker container node begins recording information when the anomalous classifier detects that the attacker is attempting to do something malicious and discretely notifies the neighboring *HoneyPeers* that an attacker is active within the network. *HoneyCollectors* are used to aggregate and collect information from each individual adversarial honeypot node and store it in the central *Adversarial Database*. All activities on the node are collected and stored with a public-key hashed *timestamp*. In our framework, the central database is a *Samba* database is used to collect *structured,unstructured*, and *semi-structured* session data to record the *adversary-honeypot-decoy* interaction. An analysis module, used to aggregate adversarial information and use that to learn about the attacker, this learned information can potentially be used to perform inference for future attacks. Figure 2 gives an illustration of our adversarial honeynet architecture.

### 3.2 Honeynet Functional Components

– **HoneyPeers** are a series of interconnected high-interaction honeypots joined in a decentralized network topology. Each HoneyPeer is an autonomous high-

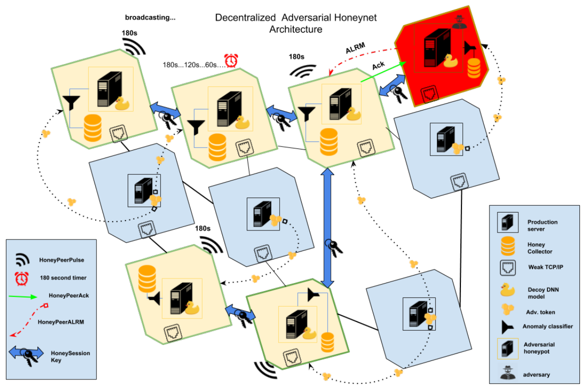

**Fig. 2.** Adversarial Honeynet Architecture

interaction honeypot contained node, with a copy of the decoy learning model $T_{decoy}$, embedded within a monitored Linux container, powered by Docker. Encrypted communication messages are passed between the nodes in order to notify adjacent nodes that an attack is occurring or has occurred. All communication is governed by our *message-passing-protocol* defined in the next section. Each *node-to-node* interaction is initiated by exchanging a *HoneySession Key*, which is used to authenticate a node's identity with each of its peers and is reused in verify future interactions. If a node should become unresponsive, it is assumed that the node has been *compromised and is infected*. In the case that a node should become infected, it can be assumed has been compromised by the adversary, in which case all neighboring nodes will sever all future communication with it, flag any local session HoneySession keys, and the infected honeypot will be cautionary labeled. Furthermore, all node-to-node interactions are securely stored and recorded in the *central adversarial database*.

– **Decoy Classifier** represents our solution for preventing the adversary from interacting with the target classifier learning model $T_{target}$, and block *transferability* from occurring by re-channeling it to the honeypot. We distribute fake *decoy* learning systems throughout the enterprise or specifically in the anterior of a production system, acting as a type of *sentinel*. In this paper, we hypothesize that legitimate users querying the learning system have no cause to interact with decoys or take notice of our adversarial honeypot. We decided to experiment with *deception-as-a-defense* using honeypot and decoys because we wanted to give the adversary a false sense of assurance, then identify and study them, and greatly reduce the rate of false-negatives *FN* violating classifier integrity.
We suspect the adversary will attack our decoy learning classifier system $T_{decoy}$ once he infiltrates the tailored honeypot container. It's purpose is to simply *simulate* and *mimic* value, in order to distract the adversary and prevent him from interacting with the legitimate target model $T_{target}$. If we

consider the adversary to be *weak*, we see that the designed adversary only has partial knowledge of the model's purpose. This means the adversary does not have possess any internal details of the architecture, hidden layers, or hyper-parameters, etc. Knowing that the adversary is in a *black-box* setting and can only access input/output gives us great leverage over him. Before the adversary launches his attack, the adversarial actor in this case is like any other regular user in the system, with no systematic knowledge of the classifier. Here, the adversary's capability to interact with the decoy model $T_{decoy}$ is represented by the *Oracle* $\hat{O}$. $\hat{O}$ represents the means for an adversary to interact with and learn from decoy model. Since the adversary wishes to produce adversarial examples $\boldsymbol{x^*}$ for a specific set of input samples $\bar{x}$, collected by querying the $\hat{O}$, and then transfer them. However, adversarial transferability can be *re-channeled* if we can switch the target model $T_{target}$ and the *Oracle* $O$ with a decoy model $T_{decoy}$ and thereupon *Oracle* $\hat{O}$, and convince the adversary that no *tampering* has occurred.

– **HoneyCollector** is the component responsible for collecting all the adversarial session information on the adversary within each of the honeypot nodes in the network, it is the *Samba* component within our system.
– **Anomaly Classifier** used to predict whether the adversary's actions inside the honeypot are considered abnormal or not. It depends on indicators, such as 1) *Number of DNN labeling requests*; 2) *execution of unusual scripts*; 3) *irregular outbound traffic from source*; 4) *sporadic DNN querying*; 5) *persistent activity on the DNN*; 6) *use of foreign synthetic data for labeling.*
– **Adversarial Tokens -** They can be thought of as a *digital* pieces 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.

### 3.3 HoneyPeer Node Inter-Communication

This section describes the message passing protocol between the nodes in the adversarial Honeynet framework. A message can only be sent and received between two HoneyPeer nodes in the network that have exchanged *HoneySession* key between them. Any message that has been received or sent spontaneously should not be accepted. A reliable message passing technology must be set in place to avoid congestion and bottleneck at one of many parts of the network. Also, all messages sent, received, and dropped are *time-stamped* and recorded within the *adversarial central database* for bookkeeping purposes.

– **HoneyPeerALRM -** a distress message indicating that host node (*Sender*) has been compromised. The message is broadcast to the nearest adversarial honeypot node in the network. The neighboring nodes (*Receivers*) are responsible for intercepting and passing the message to all neighboring nodes in the network. For obvious security concerns and as fault-resistance, another HoneyPeerALRM message is sent on behalf of the *anomalous classifier*, in the case an adversary manages to seize control of the node and hijack it after detection. Each HoneyPeerALRM message must receive an *HoneyPeerACK* to indicate that the distress HoneyPeerALRM message has been received and acknowledged. Failure to reply might indicate one or several neighboring nodes have also been compromised. To add, nodes should not receive unsolicited HoneyPeerALRM reply messages from other adversarial nodes, as this may indicate malicious misrepresentation.

- **HoneyPeerAck -** this is a message sent corresponding to each HoneyPeer-ALRM message sent on behalf of the node. A HoneyPeerACK indicates that the distress HoneyPeerALRM message has been received and confirmed by the endpoint node. Failure to receive and acknowledge one ore more *Acks* might indicate that one or all the surrounding neighboring nodes have been compromised. Also, nodes should not receive unsolicited HoneyPeerALRM reply messages from other adversarial nodes.
- **HoneyPeerSafePulse -** Periodically, a honeypot node will send a *pulse* indicating that it is still active and part of the decentralized network, and not compromised. If the node neighboring it does not reply in 180 seconds with an *HoneyPotSafeAck* response, it is assumed that the node has been compromised.
- **HoneyPeerSafeAck -** A confirmation message sent to indicate that the node is active. After 3 consecutive (60 second interval) *no replies*, it can be assumed that either the receiving node is down or has been compromised, in which case, all neighboring nodes will sever all communication with it, purge any HoneySession keys, and the infected honeypot will be labeled as an *InfectedPeer*.
- **HoneySession Key -** An adversarial session key is exchanged between two HoneyPeer nodes. This HoneySession Key is exchanged at the beginning of a *node-to-node* interaction and will be used an authentication method in future *node-to-node* communications.

## 3.4 Attracting the Adversary

**Adversarial Honey-tokens** we extended the honeybit token generator in [1] to create the *adversarial honeytokens* generator, which acts as an automatic monitoring system that generates adversarial deep learning related tokens. It is composed of several components and processes. In order to understand how the system functions, one must have an understanding of the individual operative components and processes. The following points offer an insight into how the system functions used to create token and decoy digital information to bait the adversary:

- **Baiting the Attacker -** in order for the digital tokens generated by the application to bait the attacker successfully they should have the following properties: 1) be simple enough to be generated by the adversarial honey-tokens application, 2) difficult to be identified and flagged as a bait token by the adversary, 3) sufficiently pragmatic to pass itself as a factual object, which makes it difficult for the adversary to discern it from other legitimate digital items. The purpose of these monitored (and falsified) resources is to persuade and lure the adversary away from the target DNN model $T_{target}$, and bait him to instead direct his attack efforts towards a decoy model $T_{decoy}$ residing within the honeypot trap. The goal here is to allow the adversary's malicious behavior to compromise the hoaxed model, preventing the adversarial examples transferability to the $T_{target}$ model from occurring, and forcing the attacker to reveal his strategies, in a controlled environment. The biggest challenge associated with designing these tokens is adequate camouflaging to mimic realism, to prevent being detected and uncloaked by the adversary.

– **Adversarial Token Configuration -** the configuration of the adversarial honeypot generator occurs within the *.yaml* markup file (hbconf.yaml). Here, the administrator sets the honeypot decoy *host IP address*, *deployment paths*, and *content format.* The configuration file, through the path variables, set where the tokens will be leaked inside the operating system, offering by that a large degree of freedom. Also, the administrator can customize the individual file tokens, as well as the general honeytokens and the adversarial machine learning tokens added. As mentioned, this file allows the building of several types of tokens. The first type of tokens are the *honeyfiles*, which include *txtmail*, *trainingdata*, and *testingdata.* These type of tokens are text-based and derive their formatted content from the template files stored in the templates folder. The second type of tokens include network honeybits, which include fake records deployed inside the UNIX configuration file or any arbitrary folder. The latter include general type tokens such *ssh, wget, ftp, aws, etc,* These tokens usually consist of an IP, Password, Port, and other arguments. The third type of tokens deployed are the custom honeytokens which are deployed in the bash history; these tokens are much more interesting since they take any structure or format the defender desires.

– **Token Leakage -** the most dominant feature of the adversarial honeytoken generator is its ability to inconspicuously implant artificial digital data (credentials, files, commands, etc) into the productions server's file system. The embedding location can be set inside the *.yaml* configuration file (hbconf.yaml) using the PATHS: *bashhistory, awsconf, awscred* and *hosts.* After the defender compiles and builds the adversarial tokens they are stealthily deployed at set path / locations within the designated production server's operating system. There, the tokens reside until they are found and accessed by the adversary. The Docker container at this point records intelligence on the attacker's interaction with the token.

– **Docker to Monitor the Adversary Access -** Docker was selected since it provides a free and practical way to contain application processes and simulate file system isolation, where the adversarial tokens application image will be run. In our defense framework, the numerous production servers not open to the public domain will be reserved for adversarial research to capture intelligence and analyze attacks. They will open via an exposed TCP/IP port open to the public, with weak non-privileged access points. The docker container will act as the *sandbox*, acting as entire layer to envelop the honey-token application image. Using the insight gained from the adversaries later lured to the honeypots will be used study emergent adversarial strategies, input perturbations and discovering techniques used by adversaries in their exploits. Docker will create a new container object for each new incoming connects and set up a *barrier* represented as the sandbox. An unsuspecting attacker that connects to the container and finds the tokens is presumably lured to the honeypot containing the decoy DNN model $T_{decoy}$. If the adversary decides to leave, he is already keyed to that particular container using his IP address, which connects him to the same container if he decides to disconnect and then reconnect.

– **Adversarial Token Generation -** through the extended adversarial token framework we compile the tokens using *go build* command. The following are only some of the tokens that can be generated using the adversarial honeytokens framework: 1) *SSH token*, 2) *host configuration token*, 3) *ftp token*, 4) *scp token*, 5) *rsync token*, 6) *SQL token*, 7) *AWS token*, 8) *text-mail token*, 9) *training-data token*, 10) *testing-data token*, 11 ) *comment tokens*, 11) *SSH password token*, 12) *start-cluster node token*, 13) *prepare DNN model token*, 14) *train DNN model token*, 13) *test DNN model token* and 14) *deploy DNN model token.*

## 3.5 Detecting Adversarial Behavior

One of the greatest challenges in this paper was deciding how to adequately detect, classify and label adversarial behavior as malicious. Not to mention building the actual classification model that would be responsible for doing so would have been a great undertaking on its own. However, there were other practical detection methods at our disposal, such as using signature-based detection to compare an object's behavior against a *blacklist*, and anomaly-based detection to compare an object against a *white-list.* We chose to lean towards the former method (white-list) over blacklisting since we did not have reliable adversarial data that could have been used to generate a signature to fingerprint a potential adversary. White-list detection works best when attempting to detect entity behavior that falls out of anticipated and well-defined user actions, such as *over-querying the DNN model*, or *causing a sudden decline in the classification model performance.* White-list based anomaly detection fits perfectly into our defense framework since we can characterize any pattern of activities deviating from the norm as an intrusion. The latter is in our favor since we are trying to detect actions to exploit the classifier which are novel in nature.

## 3.6 Adversarial Behavior

In order detect adversarial anomaly behavior, we have summarized a list of adversarial actions and indicators that may signal an *an-out-of the-ordinary* on the learning model. We will later use this indicators to build our white-list security rules. The following are some of those indicators:

- **Persistent DNN Querying -** while normal (non-adversaries) users will be querying the DNN $T_{decoy}$ model with 1 or 2 queries per session, the adversary will be sending hundreds, if not thousands per session. All this in effort to build his synthetic training dataset $S_p$, the adversary will need to continuously collect training data, augment it and gradually train his substitute adversarial model *F(*$S_0$*)*. Repetitive queries $\tilde{Q}$ from the same source user within a set unit of time might indicate the adversary is *query-thrashing* the DNN model for labels $(x^{'}, y^{'})$. The latter could be a possible indication of an adversarial attack on the prediction model.

- **Spontaneous DNN Activity -** in order for the adversary to craft adversarial examples $\boldsymbol{x^*}$, he will need to collect an initial set of labels $S_0$ from labeling $(x^{'}, y^{'})$. Then, he needs to build a substitute training model $F$ that mimics the learning mechanism inherent in the decoy model $T_{decoy}$. naturally, collecting enough sample labels to accurately train the model $F$ requires a large number of queries $\tilde{Q}$ solicited from the *Oracle* $\tilde{O}$. Consequently, in order to avoid raising suspicions, the adversary will try to build this initial substitute model training set $S_0$, as quickly and discretely as possible. The latter could be a possible indication of an adversarial attack on the prediction model. This is true since a few queries is within normal user behavior, who have no malicious intent in mind. But spontaneously querying the oracle falls out of normal activity.

- **High number of DNN Labeling Requests -** an abnormally high number of query requests to the *Oracle* $\tilde{O}$ is not normal either. Let us not forget, that training of the substitute model *F(*$S_0$*)* is repeated several times in order to increase the DNN model accuracy and similarity to $T_{decoy}$. With each new substitute training epoch $e$, the adversary returns to $\tilde{O}$ and queries to augment (enlarge) the substitute model training set $S_0$ produced from labeling. This will produce a large training set with more synthetic data for training. With the correct model architecture $F$, the enlarged dataset is used to prototype the models decision boundaries separating the classification regions.

- **Sudden Drop in Classification Accuracy -** building on the above and as discussed in Section 2, our designed adversary seeks to cause a misclassification attack on the target decoy model $T_{decoy}$, by inserting malicious input in the testing phase. Because of this, an input unrecognizable to the model's discriminate function can be classified with high-confidence *(false positive)*, and an input recognizable to the model can be classified with low-confidence *(false negative)*, violating the integrity of the model. Other factors may influence a drop in accuracy, such as a poor learning or added bias in the data. This does not normally occur in a production environment, which indicates that our classification model is under attack.

- other known indicator are more network related, such as *execution of unusual scripts alongside the DNN*, *Irregular outbound traffic or source*, *any sensitive or privileged path accessed during the interaction*, and *any spawning of suspicious child process.*

## 4 Related Work

The literature review below focuses directly on the concept of defending against adversarial examples, aimed at misleading the classifier. Most of the known defense methods are mainly based on data pre-processing and sanitation techniques, employed during the training phase of DNN model preparation. Pre-processing and sanitation typically mean influencing the effect that sample training-set data, $X$, has on neuron weights of the underlying DNN model, by distinguishing and filtering out malicious perturbations, inserted by an adversary that may mislead and/or confuse the classifier causing a misclassification or violation of model integrity. Other notable work in this section focus on the role of cyber-security defense through means of deception, specifically with the use of *decoys* and fake entities to deceive the attacker. Our challenge here lays in constructing a secondary-level of protection and defense, designed not to replace known adversarial defense techniques, but to supplement and reinforce existing ones, with the use of adversarial deception re-enforcing the application perimeter.
[27] focuses on addressing the lack of efficient defenses against adversarial attacks that undermine and then fool deep neural networks (DNNs). The need to tackle this issue has been amplified by the fact that there is no unified understanding of how or what makes these DNN models so vulnerable to attacks caused by adversarial examples. The authors propose an effective solution which focuses on reinforcing the existent DNN model and making it robust against 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 a bounded ReLU activation function,$f_R(x) \rightarrow y$, in the DNN architecture to stabilize the overall model prediction ability. The second is based on augmented *Gaussian* data for training. Defenses based on data augmentation improve generalization since they consider both the true input and its perturbed version. The latter enables a broader range of searches in the input, then say adversarial training, which is limited in its partial of the input, causing it to fall short. The result of applying both strategies results in a much smoother and more stable model, without significantly degrading the model's performance or accuracy.
Work in [8] is the most relevant academic paper, with regards to motivation and stimulus for the purpose of developing our proposed auxiliary defense technique, using honeypots. The authors in [8] propose a training approach aimed at building adversarial-resistant black-box learning systems against adversarial perturbations, by blocking transferability. The proposed method of training, called *NULL-labeling* works by evaluating input $\boldsymbol{x}$ and lowers confidence on the true label $y$, if $\boldsymbol{x}$ is suspected to be perturbed and rejecting it as invalid input. The

criteria on which the method evaluates $\boldsymbol{x}$ is if it spans out of the training-data data distribution area. The training method smoothly labels, filters out, and discards invalid input (NULL), which does not resemble training-data. This is to prevent from allowing it to be classified into intended target label. The ingenuity of this approach lies in how it is able to decisively distinguish between clean and malicious input. NULL labeling proves its capability in blocking adversarial transferability and resisting the invalid input that attempts to 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 true label, all while maintaining prediction accuracy.
in [21], a training approach for combating adversarial examples and fortifying the learning model. The authors propose this defense technique in response to adversarial examples, with their abnormal and ambiguous nature. The authors argue that model adversarial training still makes 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 concentrated in the classification region that has classified inputs correctly and has become resistant against malicious input perturbations that aim to exploiting and violating model integrity. This method is designed to avoid instability brought about by adversarial examples, which work by *pushing* the misclassified samples across the decision boundary into incorrect classes. This training method achieves good results on DNNs with two distinct datasets, and has low computational cost while maintaining classification accuracy for both sets.
In [2], the authors suggest a framework that actively and purposefully leaks digital entities into the network to deceive adversaries and lure them to a honeypot that is covertly monitors, tracks token access, and records any new adversarial trends. In a period of one year, the monitored system was compromised by multiple adversaries, without being identified as a controlled decoy environment. 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 fake data to deceive, attract, and fool an adversary. The authors also 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. The defender's goal is to learn as much as possible about the attacker. The message that the authors try to convey is that as the threat of adversarial attacks increases, so will the need for novelty in the defense approaches to combat it.
Work in [19], serves as an examination of the concept of *fake entities* and digital tokens, which my framework partially relies upon. Fake entities, although primitive, are an attractive asset in any security system. The authors suggest fake entities could be *files, interfaces, memory, database entries, meta-data, etc.* For the authors, these inexpensive, lightweight, and easy-to-deploy *pawns* are

as valuable as any of the other security mechanisms in the field, such as firewalls or a packet analyzers. Simply, they are digital objects, embedded with fake divulged information, intended to be found and accessed by the attacker. The authors advocate that operating-system based fake entities are the most attractive and fitting to become decoys, due to the variety of ways the operating system interface can be configured and customized. Once in possession of the attacker, the defender is notified and can begin monitoring the attacker's activity. Later in this work, the authors implement a framework that actively leaks credentials and leads adversaries to a controlled and monitored honeypot. However, the authors have yet to build a functioning 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. Among other interesting work that served as motivation for this paper include: utilizing honeypots in defense techniques, such as design and implementation of a honey-trap [5]; deception in decentralized system environments [22]; and using containers in deceptive honeypots [11]. Our approach using honeypots, does not seek to replace any of the existing methods to combat adversarial examples in a black-box attack context. However, it can be used effectively as an auxiliary method of protection that strengthen existing defense methods in production systems.

## 5 Conclusions

In this paper, we have discussed adversarial transferability of malicious examples, and proposed a defense framework to counter it, using deception derived from existing cyber-security techniques. Our approach is the first of its kind to use methods derived from cyber-security deception techniques to combat adversarial examples. We have shown it to be possible to use deception to prevent an adversary from mimicking a target model's classification behavior, if we successfully re-channel adversarial transferability. We have also presented a novel defense framework that essentially lures an adversary away from the target model, and blocks adversarial transferability, using various deception techniques. We proposed presenting the adversary with an infeasible amount of computational with no useful outcome or benefit. This can be accomplished by presenting the attacker with a hard *non-convex* optimization problem, similar to the one used for generating adversarial samples. Our framework allows the adversary to transfer these examples to a remote decoy learning model, deployed inside a high-interaction-honeypot.

## References

1. Adel Karimi: honeybits. https://github.com/0x4D31/honeybits, accessed on March 27, 2019
2. Akiyama, M., Yagi, T., Hariu, T., Kadobayashi, Y.: HoneyCirculator: Distributing credential honeytoken for introspection of web-based attack cycle. International Journal of Information Security 17(2), 135–151 (Apr 2018)

3. Carlini, N., Wagner, D.: Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods. In: Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security (AISec '17). pp. 3–14 (2017)
4. Dowling, S., Schukat, M., Melvin, H.: A ZigBee honeypot to assess IoT cyberattack behaviour. In: Proceedgings of the 2017 28th Irish Signals and Systems Conference (ISSC). pp. 1–6 (Jun 2017)
5. Egupov, A.A., Zareshin, S.V., Yadikin, I.M., Silnov, D.S.: Development and implementation of a Honeypot-trap. In: Proceedings of IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering. pp. 382–385 (2017)
6. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial esamples. International Conference on Learning Representations, 2017 p. 11 (2015)
7. Guarnizo, J.D., Tambe, A., Bhunia, S.S., Ochoa, M., Tippenhauer, N.O., Shabtai, A., Elovici, Y.: SIPHON: Towards Scalable High-Interaction Physical Honeypots. In: Proceedings of the 3rd ACM Workshop on Cyber-Physical System Security (CPSS '17). pp. 57–68 (2017)
8. Hosseini, H., Chen, Y., Kannan, S., Zhang, B., Poovendran, R.: Blocking Transferability of Adv. Examples in Black-Box Learning Systems. arXiv:1703.04318 (2017)
9. Huang, L., Joseph, A.D., Nelson, B., Rubinstein, B.I., Tygar, J.D.: Adversarial Machine Learning. In: Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence (AISec '11). pp. 43–58 (2011)
10. Irvene, C., Formby, D., Litchfield, S., Beyah, R.: HoneyBot: A Honeypot for Robotic Systems. Proceedings of the IEEE 106(1), 61–70 (Jan 2018)
11. Kedrowitsch, A., Yao, D.D., Wang, G., Cameron, K.: 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). pp. 15–22 (2017)
12. Kurakin, A., Goodfellow, I.J., Bengio, S.: Adversarial examples in the physical world. International Conference on Learning Representations (ICLR) p. 14 (2017)
13. Lihet, M.A., Dadarlat, V.: How to build a honeypot System in the cloud. In: Proceedings of the 2015 14th RoEduNet International Conference - Networking in Education and Research (RoEduNet NER). pp. 190–194 (Sep 2015)
14. Liu, Y., Chen, X., Liu, C., Song, D.: Delving into transferable adversarial examples and black-box attacks. In: Proceedings of the International Conference on Learning Representations, 2017. p. 14 (2017)
15. Nguyen, A., Yosinski, J., Clune, J.: Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 427–436 (2015)
16. Papernot, N., McDaniel, P., Wu, X., Jha, S., Swami, A.: Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks. In: Proceedings of the 2016 IEEE Symposium on Security and Privacy (SP). pp. 582–597 (May 2016)
17. Papernot, N., McDaniel, P., Goodfellow, I.: Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples. arXiv:1605.07277 [cs] (May 2016)
18. Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z.B., Swami, A.: Practical Black-Box Attacks Against Machine Learning. In: Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security (ASIA CCS '17). pp. 506–519 (2017)
19. Rauti, S., Leppnen, V.: A survey on fake entities as a method to detect and monitor malicious activity. In: Proceedings of the 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP). pp. 386–390 (2017)

20. Ribeiro, M., Grolinger, K., Capretz, M.A.M.: MLaaS: Machine Learning as a Service. In: Proceedings of the 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA). pp. 896–902 (Dec 2015)
21. Rozsa, A., Gunther, M., Boult, T.E.: Towards Robust Deep Neural Networks with BANG. Proceedings of the IEEE Winter Conference on Applications of Computer Vision (WACV), 2018 (Nov 2016)
22. Soule, N., Pal, P., Clark, S., Krisler, B., Macera, A.: Enabling defensive deception in distributed system environments. In: Resilience Week (RWS). pp. 73–76 (2016)
23. Suo, X., Han, X., Gao, Y.: Research on the application of honeypot technology in intrusion detection systems. In: Proceedings of the IEEE Workshop on Advanced Research and Technology in Industry Applications. pp. 1030–1032 (Sep 2014)
24. Tramr, F., Papernot, N., Goodfellow, I., Boneh, D., McDaniel, P.: The Space of Transferable Adversarial Examples. arXiv:1704.03453 [cs, stat] (Apr 2017)
25. Xiao, Q., Li, K., Zhang, D., Xu, W.: Security Risks in Deep Learning Implementations. arXiv:1711.11008 [cs] (Nov 2017)
26. Yuan, X., He, P., Zhu, Q., Bhat, R.R., Li, X.: Adversarial Examples: Attacks and Defenses for Deep Learning. arXiv:1712.07107 [cs, stat] (Dec 2017)
27. Zantedeschi, V., Nicolae, M.I., Rawat, A.: Efficient Defenses Against Adversarial Attacks. In: Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security (AISec '17). pp. 39–49 (2017)

## 6 Appendix

### 6.1 Deep Neural Nets (DNNs)

*Deep Neural Networks* (DNNs) are a widely known machine-learning technique that utilizes $n$ parametric functions to model an input sample $\boldsymbol{x}$, where $\boldsymbol{x}$ could be an image tensor, a stream of text, video, etc. [18]. DNNs differ from conventional neural networks is the large number of (hidden) learning layers they can use, which in return allows these models to adapt to intricate features and solve complex problems. Amongst the countless uses for DNNs' is their utility in building image classification systems that can identify an object from the its intricate edges, features, depth and colors. All of that information is processed in the hidden layers of the model, known as the *deep* layers. As the number of these *deep* layers increases, so does the capability of the DNN to model and solve complex tasks. Simply expressed, a DNN is composed of a series of parametric functions. Each parametric function $f_i$ represents a hidden layer $i$ in the DNN, where each layer $i$ compromises a sequence of *perceptrons* (artificial neurons), which a processing units that can be modeled into chain sequence of computation. Each neuron maps an input $x$ to an output $y$, $f : x \longrightarrow y$, using an *activation function f($\varphi$)*. With each layer, every neuron is influenced by a parameterized *weight vector* represented by $\theta_{ij}$. The weight vectors holds the knowledge of the DNN when it comes to training and preparing the model $F$. A DNN computes and defines model a $F$ for an input $\boldsymbol{x}$ as follows [18]:

$$F(\boldsymbol{x}) = f_n(\theta_{ij}, f_{n-1}(\theta_{n-1,j}, \cdots, f_2(\theta_{2j}, f_1(\theta_{1j}, \boldsymbol{x}))))$$

## 6.2 Security of Deep Learning

In recent deep learning literature, there has been a lot of works that has focused on deploying deep neural networks in malicious environments, in which the network is potentially exposed to numerous attacks [6] [12] [26]. At the center of these threats are *Adversarial Examples.* Adversarial examples are *perturbed* or modified versions of input samples $\boldsymbol{x}$, that are used by adversaries to mislead and exploit deep neural networks, during test time, after training of the model is completed [16]. They are injected in order to circumvent the learning mechanism acquired by the DNN with the goal of misclassifying a target label. They are crafted with carefully articulated perturbations, added to the input $\boldsymbol{x} + \delta\boldsymbol{x}$, that *forces* the DNN to display a different behavior than intended, chosen by the adversary [16]. It is important to note that the magnitude of perturbations must be kept *small enough* to have a significant effect on the DNN, yet remain unnoticed by a human being. These adversarial exploitations vary in their motivation for corrupting a DNN classifier, however some of the most common incentives range from simply reducing the confidence of a target label to a arbitrary source-label misclassification [16]. Confidence reduction entails reducing the accuracy on a label $y$ for a particular input $x$ in the testing pair $(x^{'}, y^{'})$. By contrast, source label misclassification involves having the model classify an input $x$ as a chosen target label $y_{target}$, different from the original (and intended) true source label $y_{true}$. For any attack to be successful, it requires the adversary to have previous knowledge of the DNN architecture, preferably a strong one. This knowledge can perfect *white-box* attacks, partial *black-box* attacks or *blind* attacks with no adversarial knowledge. However, it is possible to attack a DNN model $F$ with limited knowledge in hand. In past work, such as [16], the attacker was able to approximate the architecture of a target model, $F_{target}$, in a black-box setting, and create a substitute training model, which was then used to craft adversarial examples that generalize on both models. These example were transferred back to target model, by way of *adversarial transferability* [16] - a very powerful property, which enables an adversary to transfer malicious examples between models to evade a target classifier model. While deep learning networks have gathered much attention in terms of capability to solve complex and hard to solve problems, there are perilous threats that can erode and inhibit their potential [25]. It is believed that deep neural networks can be exploited from these three directions:

- *Modified Training Data* - commonly known as a *causative* or *poisoning* attack, in which the adversary influences or manipulates the training data-set $\chi$, with a transformation. This modification could entail control over a small portion or an important determinant feature dimension $D_i$ in the training data. With this type of attack advance, the attacker can mislead the learner in order to produce a badly classifier, which the adversary exploits post training [9].
- *Poorly Trained DNN Models* - although considered an oversight, rather than blamed on an external adversary. A perfunctory trained DNN could be due to several reasons. Most of the time, developers credulously use DNNs prepared

and trained by others. These same DNNs could have hidden vulnerabilities ripe for exploitation, which can become easy targets for manipulation by adversaries during deployment [25].

– *Perturbed Input Image* - commonly known as *adversarial examples* [12], attackers are also known to attack DNN models, during testing, by constructing malformed input to evade the learning mechanism learned of the DNN classifier. This is known as an *evasion attack* [9]. Our paper focuses on combating the this kind of attack.

## 6.3 Adversarial Examples

As mentioned, machine-learning models are vulnerable to adversarial attacks that seek to destabilize the neural network's ability to generalize new input; which jeopardizes the security of the model. From what we learned from the authors in [9], these attacks can either occur during the training phase 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 a poorly trained model, which cannot be easily detected. These disruptive anomalies are known as *adversarial examples.* Adversarial examples are slightly perturbed versions of regular input samples normally accepted classifiers. They are maliciously designed to have the same appearance as regular input, from a human's point of view, at least. These masquerading inputs are designed to confuse, mislead, and force the classifier to output the wrong label [8], violating the integrity of the model. These examples can be best thought of as "glitches" that can fool the deep learning model. These glitches are difficult to detect and are widely exploitable, if left unattended. To better understand them, consider this example: given an input sample $\boldsymbol{x}$ classified with function C, such that $C(\boldsymbol{x}) = \ell$, producing output $\ell$, that was correctly classified by model $A(\cdot\,)$, we say the perturbed input sample $\boldsymbol{x^*}$, so that $C(\boldsymbol{x^*}) = \ell$, we say $x^{'}$ is an adversarial example of $x$ such that $A(x^{'}) = A(x)$. Classification models are considered *robust* if their classification ability is unaffected by the presence and exploits of adversarial examples. adversarial examples $\boldsymbol{x^*}$ possess an appearance similar or *close* to the original input samples $\boldsymbol{x}$. Normally used, although not the only form of measurement. This measure of *closeness* or *similarity* between the pair of original and modified input is known as the $p$-norm distance $\parallel x \parallel_p$. This degree of closeness could be $l_2$, which is the *Euclidean Distance* between two pixels in an input sample $x$, $l_\infty$, which is the absolute or max change made to a pixel in $x$, or $l_1$; which is the total number of pixel changes made to the input sample $x$ [3]. If the measure of distortion in any of the previous metrics of closeness is small, then those input samples must be visually similar to each other, which made them a prime candidate for adversarial example generation. lose to each other for adversarial transferability to be successful.

### 6.4 The Adversarial Optimization Problem

Generating adversarial examples means there is a computational cost involved. In the general case, adversarial examples are generated by solving a hard optimization problem similar to the one below [18]:

$$\boldsymbol{x}^* = \boldsymbol{x} + argmin\{\boldsymbol{z} : \hat{O}(\boldsymbol{x} + \boldsymbol{z}) \neq \hat{O}(\boldsymbol{x})\} = \boldsymbol{x} + \delta_x$$

Where $\boldsymbol{x} + \delta_{\boldsymbol{x}}$ represents the least possible amount of *noise* added to cause a perturbation, while remaining unnoticeable by humans. The adversary wishes to produce adversarial examples $\boldsymbol{x}^*$ for a specific input sample $\boldsymbol{x}$ that will cause a misclassification by the target model $T_{target}$, with a queried adversarial sample, such that $O\{\boldsymbol{x} + \delta_{\boldsymbol{x}}\} = O\{\boldsymbol{x}\}$. This misclassification proves that the classifier has been compromised, and is no longer usable. The misclassification error and drop in target label accuracy the attacker is after is achieved by adding the least amount of possible noise $\delta_{\boldsymbol{x}}$ to the input $\boldsymbol{x}$, in order to be unnoticeable by humans, but just enough to mislead the DNN. Solving for $\boldsymbol{x}^*$ is an optimization problem that is not easy to solve since it is *non-linear*, where multiple true solutions exist, and *non-convex*, where there not so easy to find. An optimization problem is considered to be *convex* if convex optimization methods can be used on the cost function $J(\theta)$, that if minimized $\min_x J_0(x)$, for the best possible and unique outcome can guarantee a global optimal solution. In convex-type problems, optimization is likely a well-defined problem here with one optimal solution or *global optimum* across all feasible search regions. On the other hand, a *non-convex* problem is one where multiple local minimums exist (solutions) exist for the cost function $J(\theta)$. Computationally, it is difficult to find one solution that satisfied all constraints. Here, optimality has become a problem, and an exponential amount of time and variables are required to find a feasible solution, where many indeed exist. By preventing the attacker from learning anything about the model $T_{target}$ in a black-box system setting; it makes it more difficult to solve this computational challenge.
In our approach, we introduce this *difficulty* by deceiving the adversary and allowing him to attempt in solving this optimization problem, as an infeasible task for a decoy model $T_{decoy}$, which has no real value. Generating these adversarial examples is already exhaustive in computational cost time, as well as approximating and training the substitute decoy model to craft the examples. And if the attacker does indeed succeed in generating these examples, it would a highly infeasible task done in vanity.

### 6.5 Impact of Adversarial Examples on Deep Neural Nets

As it is known, a machine-learning application could be in severe jeopardy if the underlying model were to fall in the hands of an adversary, with intentions on launching an attack. However, there are certain measures taken to prevent the latter from occurring. However, equally menacing, and as likely, is if an adversary were able insert an input, image or query that would bypass the model's

learning mechanism, and cause a misclassification attack, in full view of the defender. Adversarial Examples have the ability to do just that.
Deep neural nets depend on the discriminate features $X_{m,n} = (x_{1,1}, x_{1,2}, x_{1,3}, \ldots, x_{1,n})$, embedded within the image that the DNN model recognizes and learns, which it then assigns to its correct class label. However, according to [15] it was shown that the DNN models can be *tricked* and convinced that a *slightly* perturbed image or input that should otherwise be unrecognizable and consequently rejected by the neural network, can be *forced* to be generalized and accepted as a recognizable member of a class in the targeted model. The consequence of this is that many state-of-the-art machine-learning systems deployed in a real-world setting are left vulnerable to adversarial attacks, at any point in time from any user. This creates calamity, because any chosen input unrecognizable to the model can be *transformed* and classified with high confidence causing a *(false positive)*, and an input recognizable to the model can be classified with low confidence *(false negative)*, violating the integrity of a prediction model, eventually making it unusable. For instance, some of the most striking examples are in the case of audio inputs that sound unintelligible (to human), but contain voice-command instructions that could mislead the deep neural network [12]. In the case of facial recognition scenario, where the input is subtly modified with markings that a human being would recognize their identity correctly, but the model identifies them as someone else [12].

### 6.6 Adversarial Transferability

According to the authors in [24], 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 $\boldsymbol{x}$, the adversarial examples $\boldsymbol{x}^*$ generated to confuse and mislead one model $m$ can be *transferred* and used to confuse other models $n_1, n_2, n_3, ..., n_i$, that are of homogeneous or even heterogeneous classifier architectures. This mysterious phenomena is mainly due to the determining 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 [8]. 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 one hypothesis in [6], 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 authors hypothesize that the higher dimensionality of the training data example set $D$, the more likely that the sub-spaces will intersect significantly, guaranteeing the transfer of samples between the two sub-spaces [6]. According to the above hypothesis, transferability holds true between two models *as long as both models share a similar purpose or task* [17].

Knowing this, an attacker can leverage the property of transferability to launch an preemptive attack, by training a local substitute classifier model $F$ on sample testing data pairs $(x^{'}, y^{'})$, that the chosen remote target classifier $T_{target}$ were generalized on. Collecting these testing pairs can be formed into a training dataset $D_{training}$ of size $N$ of similar dimensions and content. With the latter we can produce adversarial examples $\boldsymbol{x^*}$. It is also worth noting that the success rate of transferability varies depending on the type of remote target classifier the examples $\boldsymbol{x^*}$ are being transferred to. These modified examples can then be transferred to the target classifier. Hence, the same perturbations that influence model $n$ also effect model $m$. 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 [18], which is the same attack we also used for our designed adversary. This transfer property is an anomaly, and creates an obstacle in the face of deploying and securing machine-learning services on the cloud, enabling exploitation and ultimately attacks on black-box systems [24], as we'll see in the coming sections.

## 6.7 Black-Box Learning Systems

To explain a black-box threat model, we start by describing the term *black-box* system concept. 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)* = $y$ transforming $x$ into $y$ remaining invisible. The functionality of the black-box can only be understood by observation, which is what the attacker depends on to begin his attack. The black-box threat model is by extension a black-box system. In our paper, we are attempting to prevent the attacker from polluting the target classifier $T_{target}$, by blocking transferability and access to the target model to change the prediction on the class label $y$. Here, we consider the adversary to be *weak* with limited knowledge, as in he can only observe the inputs inserted and outputs produced, while possessing little knowledge of the classifier itself. The adversary possesses 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 need not know the internal details of the system to exploit and compromise it [18].

Generally, in order to attack the model, in a black-box learning setting, the adversary attempts to generate adversarial examples, which are then transferred from the substitute classifier $F$ to the target classifier $T_{target}$, in an effort to successfully distort the classification of the output labels [8]. The intention 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 latter purpose, the attacker continuously updates the substitute learning model and queries the target classifier (represented by the 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*. Querying the *Oracle* represents the only capability which the attacker possesses. Querying the *Oracle* $O$ for input $\boldsymbol{x}$, which represents the only capability available to the attacker, as in the black-box model no access to the Oracle internal details is possible [18]. The goal of the adversary is to produce a *perturbed* version of any input $\boldsymbol{x}$, known as an *adversarial sample* after modification, denoted $\boldsymbol{x}^{*}$. This represents an attack on the integrity of the classification model (oracle) [18]. What the adversary attempts to do is solve the following optimization problem to generate the adversarial samples, as seen below:

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

The adversary must able to solve this optimization problem by adding a perturbation at an appropriate rate with $\delta_{\boldsymbol{x}}$, to avoid human detection. The magnitude $\varepsilon$ of the rate must be generated in such a way with the least perturbation possible in $\delta_{\boldsymbol{x}}$ to influence the classifier, as well remain undetected by a human [18]. This is considered a hard optimization problem, since finding a minimal value to $\delta_{\boldsymbol{x}}$ is no trivial task. Further more, removing knowledge of the architecture and training data makes it difficult to find a perturbation that satisfied the condition for successful adversarial examples secretion, where $O\{\boldsymbol{x} + \delta_{\boldsymbol{x}}\} = O\{\boldsymbol{x}\}$ [18].

## 6.8 Transferability and Black-Box Learning Systems

Adversarial Transferability is critical for black-box Attacks, to say the least. In fact black-box systems are dependent on its success. In [25], it is suggested that the adversary can build a substitute training model $F$ with synthetic labels $S_0$ collected by observing the labeling of test samples by the *Oracle* $O$, despite the DNN model and dataset being inaccessible. The attacker can then build a substitute model $F$ from what he learns from $O$. The attacker will can then craft adversarial samples that will be misclassified by the substitute model $F$ [16]. Now that the attacker has approximated the knowledge of the internal architecture of $F$, he can use it to construct. For as long as adversarial transferability holds between $F(S_0)$ and $T_{target}$. adversarial examples misclassified by $F$ will be misclassified by the target as well. In our paper, we find a way to re-channel adversarial transferability and prevent an attack. We plan to accomplish the latter via *deception*. It was Papernot in [18] [17], who proposed that transferability can be used to transfer adversarial examples from one neural network to the other that share a common purpose or task, yet are dissimilar in network architecture. Transferability is essential for the success of black-box attacks on deep neural nets, which is due to the limitations imposed on the adversary, such as lack of architecture, model and training dataset knowledge. Even with limited knowledge, the adversary with the aid of the transferability property in the adversary's armaments, the adversary can train a substitute model and generate transferable examples, then transfer them to the unprepared target model, making the victim's trained model vulnerable to attack [26]. There has been much work focused on the abilities possessed by adversarial examples, and its ability to transplant itself between machine-learning techniques (DNN, CNN, SVM, etc.). Work, namely in [3] [14] [18], all reached the same conclusion - adversarial examples will transfer across different models trained on different dataset implementations, with different machine-learning techniques.

## 6.9 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. A honeypot is meant to appear and respond like a real system, within a production environment. However, the data contained within the honeypot is both falsified and spurious, or better understood as *fake*. 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 contains no real data and therefore poses no threat on the production environment [13] [23]. As mentioned, honeypots can be deployed with fabricated information, this can be an attractive target to outside attackers, and with the correctly engineered characteristics can be used to re-direct attackers towards decoy systems and away from critical infrastructure [7]. 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 [4]. 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* systems 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 can potentially capture the full spectrum of attacks launched by adversaries within the system. It 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.