2022-05-01

Game-Theoretic Deception Modeling for Distracting Network Adversarie

Mohammad Sujan Miah
The University of Texas at El Paso

GAME-THEORETIC DECEPTION MODELING FOR DISTRACTING NETWORK ADVERSARIES

MOHAMMAD SUJAN MIAH

Doctoral Program in Computer Science

APPROVED:

Christopher Kiekintveld, Ph.D., Chair

Munindar P. Singh, Ph.D.

Mahmud Shahriar Hossain, Ph.D.

Deepak K. Tosh, Ph.D.

Stephen Crites, Ph.D.
Dean of the Graduate School

to my

SISTER and FAMILY

with love

GAME-THEORETIC DECEPTION MODELING FOR DISTRACTING NETWORK ADVERSARIES

by

MOHAMMAD SUJAN MIAH

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

Department of Computer Science

THE UNIVERSITY OF TEXAS AT EL PASO

May 2022

Acknowledgements

I would like to express my deep gratitude to Dr. Christopher Kiekintveld for all his help and support. Without your consistent encouragement and advice, I would not be able to complete my Ph.D. Thank you, Professor, for your belief in me. I would also like to express my gratitude to the other members of my committee, Dr. Munindar P. Singh, Dr. Mahmud Shahriar Hossain, and Dr. Deepak K. Tosh, for their constant invaluable suggestions, wise advice and guidance, which helped me complete my goal.

I am thankful to my lab mates Porag Chowdhury and Anjon Basak for their support in many ways. I have learned a lot from working with my other lab mates: Marcus Gutierrez, Anthony Ortiz, Nazia Sharmin, Oscar Veliz, and Alonso Granados. Thank you all.

Omkar Thakoor, a collaborator at the University of Southern California, deserves special thanks. I feel fortunate to have been working with the University of Houston, thanks to Dr. Aron Laska and Shanto Roy. I have learned a lot working with North Carolina State University. I am grateful to Dr. Munindar P. Singh, Dr. William Enck, Mu Zhu, Iffat Anjum, and Ruijie Xi. In my honeyflow optimizing game models, I discussed deception architecture and honeyflow generation concepts as motivation that we accomplished together. Thank you all. Also, I would like to give a special thanks to Dr. Palvi Aggarwal.

Finally, I want to thank my friends who have been always on my side. I want to express my gratitude to my wife Fahima and other family members for their unwavering support.

Abstract

In this day and age, adversaries in the cybersecurity space have become alarmingly capable of identifying network vulnerabilities and work out various targets to attack where deception is becoming an increasingly crucial technique for the defenders to delay these attacks. For securing computer networks, the defenders use various deceptive decoy objects to detect, confuse, and distract attackers. By trapping the attackers, these decoys gather information, waste their time and resources, and potentially prevent future attacks. However, we have to consider that an attacker with the help of smart techniques may detect the decoys and avoid them. One of the well-known challenges in using decoys is that it can be difficult to design effective decoys that are hard to distinguish from real objects, especially against sophisticated attackers who may be aware of the use of decoys. Both real and decoy objects have observable features that may give the attacker the ability to distinguish one from the other. One way for a defender to enhance a decoy's effectiveness is to modify a few features of either the real or fake objects. But, such information manipulation or system modification for the defender needs to be cost-effective. Game-theoretical models are often useful to analyze strategic interactions between agents to find the best decision-making solutions. In this thesis, I study some game-theoretic and adversarial machine learning models to determine optimal strategies for the defender and focus on employing decoys to prevent security threats.

The first game model I work to design practical decoy objects that can fool a sophisticated attacker. This model allows us to investigate many aspects of how a defender should optimize efforts to conceal deceptive objects, which can be applied to honeypots, disguising network traffic, and other domains. Furthermore, its theoretical foundation provides the benefits and limitations of adversarial learning methods for generating deceptive objects. In our model, we allow the defender to modify either the real or fake object that renders objects indistinguishable for the attacker thus, improve deception noticeably. By

using this model, we seek to capture some key aspects of cyber deception that are missing from other game-theoretic models. In particular, we focus on whether the defender can design convincing decoy objects and the limitations of deception if some discriminating features of real and fake objects are not easily maskable. To my knowledge, this the first model that introduces a two-sided deception technique to mislead the attackers. However, an important element to take into account for the use of two-sided deception is cost. Here we show, in some cases, deception is either unnecessary or too costly to be effective. The deception level mainly relates to attackers' sophistication, wherein naïve attackers are easy to deceive even with a low-cost strategy. This game model provides a new and more nuanced way to consider the quality of various deception strategies but strives to solve large and complex two-sided feature deception problems. To further develop and scale the model, we use the Adversarial Machine Learning (AML) approach that can generate fake samples that look like real samples and real samples that look like fake samples when the feature space is complex and large. The technique can also be used as a robust classifier for the binary classification problem in a dynamic learning environment. We also present the empirical analysis of the AML algorithm and discuss some possible use cases of our model.

The next game-theoretic model I design to use deceptively crafted honey traffic to confound the knowledge gained by an adversary through passive network reconnaissance. This model characterizes how a defender should deploy honey traffic in the presence of an sophisticated attacker and finds the optimal strategy for deploying honey flows that are fast enough to be used for realistic networks. These optimal defender strategies deter an attacker from acting on the existence of real vulnerabilities found in network traffic. Our proposed model balances cost and benefit trade-offs, but can still be solved quickly enough to be used in a complex network environment. We show that the strategic optimization benefit is the highest when the cost of producing honey flow is reasonable, which is the most likely real application scenario, and the network defender should generate more honey flows to cover the highly valued vulnerabilities. We extend the current game model

by addressing that the attack distributes beliefs over various types of honey traffic, reflecting the quality of the honey flows, indicating how hard it is for the attacker to distinguish. In addition, the attacker needs to pay a cost to attack. This model further captures how the honey traffic quality impacts attacks decision-making strategies. We show that high-quality honey traffic makes it harder for the attacker to distinguish between real and honey traffic.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In recent years, the advent of Artificial Intelligence (AI) has rendered machines to become smarter, increasingly capable and independent, influencing how we live and work. One such example of developing AI algorithms is self-driving cars showing significant promise in terms of safety and efficiency. The impact of Artificial intelligence is expanding in numerous applications in various fields such as decision-making, economics, healthcare, verification, security control, to name a few, making our life more comfortable and convenient. However, AI's advancement though beneficial, has raised some concerns too; we are slowly witnessing the machines taking over jobs and thus putting humans out of work. Furthermore, AI becomes an additional threat to our security. The human race is now not only battling against human adversaries but also intelligent agents. But let's not be hasty; artificial intelligence is still in its infancy. AI algorithms are incredibly limited and vulnerable in learning and decision-making in the dynamic and unknown environment instead of true independent agents who can do these on their own.

Game theory, a branch of AI, studies the strategic interactions between different agents to find the best decision-making solution in a context. It helps to analyze many situations we may face in our lives, especially when equipped with limited resources to confront attackers in an adversarial setting where the best outcome depends on optimal resource allocation. Many game models are popular nowadays to analyze and solve multi-agent interaction problems, such as Normal Form Game, Stackelberg Games, Bayesian Game, Extensive-Form Games, etc. In this thesis, I focus on using decoy resources to ensure maximum cybersecurity and investigate different game-theoretic models to determine the defender's optimal deceptive strategies while interacting with attackers.

With the widespread growth of computer networks, the attackers are becoming more sophisticated over time, and standard countermeasures are no longer enough to provide network security. But deceiving the attackers can improve cybersecurity by introducing uncertainty in a computer network. Cyber deception methods use deceptive decoy objects like fake hosts (honeypots), fake network traffic, files, and even user accounts to counter attackers in a variety of ways [1, 2, 3]. For instance, a honeypot is installed in a computer network to create confusion for attackers, make them more hesitant and less effective in executing further attacks. Also, it can help to gather information about the behavior and tools of various attackers. Through it, the defenders can detect malicious activities and actors in the network. However, deceptive objects' effectiveness boils down to how closely these objects can be designed to resemble the real objects, so they are not easily identified and avoided. A critical problem in deploying a decoy object is that both real host and decoy object may have observable features that may give the attacker the ability to distinguish one another.

This thesis will present the first game-theoretic model called *Honeypot Feature Selection Game* (HFSG). It finds the optimal decisions for the defender who is trying to disguise the identity of real and honeypots so that the attacker can not reliably distinguish between them. Generally, honeypots are designed in such a way that the observable characteristics simulate the real host properties. However, creating realistic honeypots is highly expensive, sometimes impossible. Also, deceiving the attackers by using and adapting honeypots (one-sided deception) is not adequate to secure a computer network. Therefore, my first research question is:

Q1.1. How to model a framework by utilizing features deception to make effective cyber decoys?

In the HFSG model, we introduce a two-sided deception approach where a defender can modify system features by paying some costs to make the deception more convinc-

ing. Unlike many models of deception, we consider the possibility that the defender can make changes to both the real and fake objects. This game model allows us to study cyber deception's strategic aspects against different adversaries under several practical assumptions where an adversary may or may not aware of the deception. Through it, we can evaluate the cost involved in such deception, their benefits, and discern the abilities of the players. Most importantly, it can answer how and when we should create decoy objects.

Our experiments show that applying deception always provide better benefit to the defender rather than no deception. The efficacy of deception is related proportionally to the number of features available to change. When objects provide more modification space to the defenders, it creates more confusion for attackers in distinguishing the real hosts and honeypots. We found that when the feature distributions become more informative, two-sided deception provides a useful advantage over one-sided deception. The analyses also show that two-sided deception is more likely to mislead naïve attackers when the feature distributions are similar between real and honeypots, but the opposite is true for a rational attacker. However, a critical problem of two-sided deception modeling is that real hosts' modification can affect network performance. Therefore, we need to be cautious in selecting the features of a real host' for change. Now, my next research question is :

Q1.2. When should we consider employing the two-sided deception strategy for concealing cyber decoys?

Generally, we avoid altering the real hosts' features and only employing one-sided deception; try to modify the honeypot to look like a real host. Also, there are some extreme cases for both real hosts and honeypots where some features may not be possible to modify at all. In this work, I try to identify those features that the defender should focus on modifying to make the deception more useful, including real hosts' features. Also, I seek to find situations where deception is not the best solution because of the high costs of creating a believable deception.

For the experiment, we set the modification costs of some features to infinity in the HFSG model. The results show that if the same feature for the real host and honeypot is unmodifiable and the feature distributions are highly dissimilar, then two-sided deception provides minimal benefit to the rational attacker. But, when the features that cannot be modified are different for the real and honeypot hosts, it is an excellent benefit for the defender to use two-sided deception. The experiment also shows that when the feature descriptions are more informative and the real features modification costs are high, the two-sided deception has a small advantage over one-sided deception. But when the real feature modifications are not too costly, the two-sided provides a noticeable improvement in defenses, even the feature distribution difference is high.

The HFSG game model allows us to evaluate the effectiveness of cyber deception under several different realistic assumptions about the costs and benefits of deception and the abilities of the players. We identify cases where deception is highly beneficial, as well as some cases where deception has limited or no value. We also show that in some cases, using two-sided deception is critical to the effectiveness of deception methods. However, one limitation of the model is that we analyzed the two-sided deception approach using a small number of features. Also, the time and memory complexities of the game model depend on the number of hosts, the number of features, feature modification options, and the amount of sampling, which makes the model grow exponentially. Therefore, the following is my next research question :

Q2. How to extend and scale the HFSG model to solve much larger two-sided problems with vast numbers of complex features?

Recently, adversarial machine learning models have shown great promise in generating deceptive objects, and the most well-known approach is Generative Adversarial Networks (GAN). The intuition of GAN is that the networks are playing a zero-sum game. Typically GANs are used in much larger problems with vast numbers of complex features and

generate fake samples without modifying real inputs. On the other hand, adding small perturbations to the original inputs could mislead machine learning models with high confidence. Therefore, we use Adversarial Machine Learning methods to solve the scaling problem of two-sided feature deception and and introduce an algorithm called TS-GAN. We model the problem as a two-player minimax game between the defender and attacker. The defender attempts to confuse the attacker by creating fake and adversarial samples, the latter of which is created by modifying a real sample in such a way that the attacker mispredict it as a fake sample. The attacker's goal is to correctly distinguish between real and fake samples to minimize his expected loss. On the other hand, the defender's strategy is to maximize the attacker's expected loss.

For experiments, We investigate the impact of varying the similarity between the HFSG and TS-GAN models. In the game model, two-sided deception has an advantage in highly informative cases, and the utility of the defender grows more as the feature distributions become more informative. We also see a similar pattern in the TS-GAN model: the attacker loss is higher in two-sided feature manipulation than in one-sided feature manipulation. Fake image quality isn't good enough in the early stages of learning, resulting in a more significant gap between real and fake samples. As a result, the attacker loss in the TS-GAN is higher. The two-sided approach can help us figure out when and how to use deception, and one conceivable application is for network traffic to be disguised as other traffic using the models. There are many reasons to disguise network traffic to look like other traffic; defenders may wish to generate fake traffic to support honeypots or conceal the properties of real traffic on their networks otherwise. Attackers also may want to make their network traffic appear similar to real traffic to avoid detection. Now, the question is:

Q3. How to deploy fake network traffic optimally to deceive an attacker in the reconnaissance phase?

In a network, the most significant threat is posed by Advanced Persistent Threat (APT) actors who use sophisticated techniques to compromise the network and remain inside for

long periods to gather valuable data by network reconnaissance. These attackers seek to stay undetected and gradually identify various network vulnerabilities from the collected data by statistical traffic analysis. Now, I am moving to another problem where we use honey flows (fake packets) to expose different fake vulnerabilities to deceive attackers in the network reconnaissance phase. In this work, I model a Normal Form Game (NFG) that characterizes how we should deploy honey traffic that potentially prevents an adversary from acting on the existence of real vulnerabilities observed within network traffic.

The game model determines the optimal strategy for deploying honey flows fast enough for realistic networks. The experiment shows that the strategic optimization value is the highest when honey flow generation costs are moderate, which is the most likely real application scenario. The results also show the network defender should create more honey flows for disguising the high valued vulnerabilities that potentially divert the attacker towards the less valued systems. However, this model does not consider the impact of honeyflow quality on attackers' decision-making strategies. Therefore, we extend the existing model for further investigation where the attacker could distribute beliefs over various types of honey traffic that reflects the quality of the honeyflows, demonstrating how difficult it is for the attacker to detect the dissimilarity. We also take into account the cost of an attacker in attacking a particular type of vulnerability. We find that the quality of honey flow significantly impacts an attacker's decision-making strategy. Honeyflows that look like real traffic make it harder for attackers to distinguish between real and fake vulnerability.

Chapter 2

Related Works

2.1 Deception and Game Theory

Cyberspace is gradually becoming more vulnerable to attackers, but deceiving them can improve cybersecurity. Decoy objects (e.g.honeypots) are deployed in computer networks to analyze malicious activities of the attackers by trapping them [4, 5]. The advance targeted attack where an attacker practices network reconnaissance can also be restrained by using deception [3]. However, long time deception is not always possible, also costly. The attackers may identify decoy objects by scanning and try to avoid interacting with them. The deception interaction between a defender and an attacker is an increasing interest of study to ensure maximum network security. For many years, the game-theoretic approaches have been very successful in finding optimal strategies for allocating deceptive resources, sending fake signals, concealing attributes of a decoy, quantifying deception, and detecting the goals of the attackers [6, 7, 8, 9, 10].

Many previous works have proposed different game models to practice deception and information manipulation. *Honeypot Selection Game* [11, 9] tried to solve a key decision-making problem of allocating honeypots to a network that maximizes security against a rational attacker. In a real-world network, all systems are not equally important, for example, a military database server is more valuable than a laptop. Therefore, the values of honeypots should not always be the most or least while deploying. This model formulated a zero-sum game to determine the defender's optimal strategy of increasing the probability that an attacker targets a honeypot rather than a real system. . The *Cyber Deception Game* (CDG) [12] computes optimal deception strategy for disguising certain characteristics of

the network hosts. This approach potentially introduces uncertainty to the network and invalidates the attacker's gathering information in the reconnaissance phase, but the model is limited to zero-sum settings. Another limitation of this CDG model is the attacker's consideration as a rational, who is aware of the deception, but this is not a general case in a real-world network. But, the CDG proposed in [13] considers both rational and naive attackers, unawares of fixed preferences over observed network hosts, and determines solutions for both. The *Cyber Camouflage Games* (CCG) [14] extended the CDG model by considering a general-sum setting with addressing uncertainties in the defender's knowledge of the attacker valuations. Determining the optimal solution of the CCG is NP-hard, but solvable by using an approximation algorithm.

Network traffic obfuscation is another way of deception to foil network reconnaissance. Encryption and adding a pad in traffic features at various levels such as ciphertext formats, stateful protocol semantics, and statistical properties are effective ways of preventing statistical traffic analysis [15, 16]. Sending fake traffic with real traffic can also manipulate an adversary's observation of a particular traffic pattern and efficiently camouflage network traffic [17]. However, this approach is usually inefficient and sometimes incurs huge network overhead. While there are many approaches used to obfuscate network traffic, the Water-filling algorithm is optimal in chaff-aided traffic obfuscation for a given chaff budget [18]. The game-theoretic model is an efficient way of finding the optimal strategy to deploy the proper amount of chaff with limited resources [19], but not enough works were done before.

Chapter 3

Background

3.1 Normal Form Game

Game theory is the study of decision-making problems where multi-agent interactions are analyzed to produce optimal outcomes. There are several types of game models: single-shot, sequential, cooperative, non-cooperative, finite, infinite, stochastic, and others. Normal form games are the most familiar type of game models for analyzing multi-agent interactions, also the most fundamental representation in game theory.

Definition 1 *(Normal Form Game [20]). A normal-form game is represented by a tuple* (N, A, u)*, where :*

- *N is a finite set of n players, indexed by i*
- $A = A_1 \times .... \times A_n$*, where* A_i *is a finite set of actions available to player i. Each vector* $a = (a_1, ..., a_n) \in A$ *is called an action profile;*
- $u = (u_1, ..., u_n)$ *where* $u_i : A \mapsto R$ *is a real-valued utility (or payoff) function for player* i*.*

Normal form games are also known as matrix-form games where an n-dimensional matrix represents an n-player game. Table (3.1) represents a matrix-form of a two-player normal form game where each row is labeled with a possible strategy for player 1 (A and B), and each column is labeled with a possible strategy for player 2 (C and D). If player 1 plays that row A and player 2 plays column C, player 1 gets payoff 1 and player 2 gets -1,

Table 3.1: Example of a normal form game

		Player 2	
		C	D
Player 1	A	$(-1,-1)$	$(-4,0)$
	B	$(0,-4)$	$(-3,-3)$

respectively. The primary purpose of modeling a multi-player game is to find the optimal strategies for each player that maximizes his expected payoff.

Definition 2 *(Mixed strategy [20]). Let (N, A, u) be a normal-form game, and for any set X let $\prod(X)$ be the set of all probability distributions over X. Then the set of mixed strategies for player i is $S_i = \prod(A_i)$.*

In a pure strategy, a player plays a single action from his available actions, and the set of strategies for each player is called a pure-strategy profile. But when a player sets some probability distribution over his available actions, it is called a mixed strategy.

Definition 3 *(Mixed-strategy profile [20]). The set of mixed-strategy profiles is simply the Cartesian product of the individual mixed-strategy sets, $S_1 \times .... \times S_n$.*

The probability of an action a_i that played under the mixed strategy s_i is denoted $s_i(a_i)$.

Definition 4 *(Best response [20]). Player i's best response to the strategy profile s_{-i} is a mixed strategy $s_i^* \in S_i$ such that $u_i(s_i^*, s_{-i}) \geq u_i(s_i, s_{-i})$ for all strategies $s_i \in S_i$.*

The best response is a strategy that determines a player's best possible outcome when the other player's strategy is given.

Definition 5 *(Nash equilibrium [20]). A strategy profile $s = (s_1, ..., s_n)$ is a Nash equilibrium if, for all agents i, s_i is a best response to s_{-i}.*

A Nash equilibrium is a solution concept in game theory where each player performs the best response to all other players to reach a stable strategy profile means, and no player would benefit by changing his strategy from that strategy profile.

3.2 Extensive Form Imperfect-Information Game

In an extensive form game, the choice nodes of a player are partitioned into information sets when the player has imperfect information about the game condition, also the nodes belong to the same information set are indistinguishable to him.

Definition 6 *(Imperfect-information Game [20]). An imperfect-information extensive form game is represented by a tuple* $(N, A, H, Z, \chi, \rho, \sigma, I)$*, where :*

- *N is a finite set of n players, indexed by i*
- *A is a (single) set of actions, where* A_i *denotes player* i^{th} *available action set*
- *H is a set of non-terminal choice nodes*
- *Z is a set of terminal nodes, disjoint from H*
- $\chi : H \mapsto 2^A$ *is the action function, which assigns to each choice node a set of possible actions*
- $\rho : H \mapsto N$ *is the player function, which assigns to each non-terminal node a player* $i \in N$ *who chooses an action at that node*
- $\sigma : H \times A \mapsto H \cup Z$ *is the successor function, which maps a choice node and an action to a new choice node or terminal node such that for all* $h_1, h_2 \in H$ *and* $a_1, a_2 \in$*, if* $\sigma(h_1, a_1) = \sigma(h_2, a_2)$ *then* $h_1 = h_2$ *and* $a_1 = a_2$
- $u = (u_1, ..., u_n)$*, where* $u_i : Z \mapsto \mathbb{R}$ *is a real-valued utility function for player* i *on the terminal nodes* Z

- $I = (I_1, ..., I_n)$, *where* $I_i = (I_{i,1}, ..., I_{i,k_i})$ *is an equivalence relation on (i.e., a partition of)* $\{h \in H : \rho(h) = i\}$ *with the property that* $\chi(h) = \chi(h)$ *and* $\rho(h) = \rho(h)$ *whenever there exists a j for which* $h \in I_{i,j}$ *and* $h \in I_{i,j}$

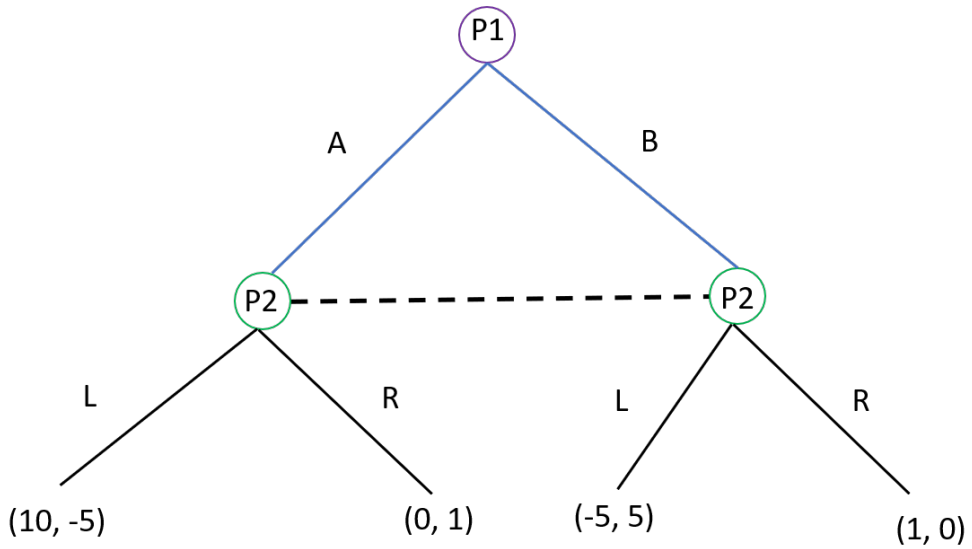


Figure 3.1: Imperfect-information extensive form game

Figure (3.1) represents an example of 2-player extensive form game with imperfect information. In this game, the top most choice node is the only information set for player 1 where A and B are the action choices. Also, player 2 has one information set, which includes the bottom choice nodes. Here, player 2 is uncertain whether player 1 chose A or B while choosing between L and R.

Definition 7 *(Pure strategies [20]). Let* $G = (N, A, H, Z, \chi, \rho, \sigma, I)$ *be an imperfect information extensive-form game. Then the pure strategies of player i consist of the Cartesian product* $\prod_{I_{i,j} \in I_i} \chi(I_{i,j})$

In an imperfect information game, a pure strategy for a player is any action from all possible actions in each player's information set. In Figure (3.1), the pure strategies of player 1 are {A, B} and of player 2 {L, R}.

Definition 8 *(Perfect recall [20]). Player i has perfect recall in an imperfect-information game G if for any two nodes h, h' that are in the same information set for player i, for any path $h_0, a_0, h_1, a_1, h_2, ..., h_n, a_n, h$ from the root of the game to h (where the h_j are decision nodes and the a j are actions) and for any path $h_0^i, a_0', h_1', a_1', ..., h_m', a_m',\ h'$ from the root to h' it must be the case that:*

1. $n = m$
2. *for all* $0 \leq j \leq n,\ h_j$ *and* h_j' *are in the same equivalence class for player* i *and*
3. *for all* $0 \leq j \leq n$, *if* $\rho(hj) = i$ *(i.e.,* h_j *is a decision node of player i), then* $a_j = a_j'$

G is a game of perfect recall if every player has perfect recall in it

In a perfect recall game, no player forgets any information he knew about moves made so far, precisely he remembers all information of his own moves.

Definition 9 *(Sequential equilibrium [20]). A strategy profile S is a sequential equilibrium of an extensive-form game G if there exist probability distributions $\mu(h)$ for each information set h in G, such that the following two conditions hold:*

- $(S, \mu) = lim_{n \to \infty}(S^n, \mu^n)$ *for some sequence* $(S^1, \mu^1), (S^2, \mu^2), ...$,*where* S^n *is fully mixed, and* μ^n *is consistent with* S^n *(in fact, since Sn is fully mixed,* μ^n *is uniquely determined by* S^n*); and*
- *For any information set h belonging to agent i, and any alternative strategy S_i' of i, we have that*

$$u_i(S|h, \mu(h)) \geq u_i((S', S_{-i})|h, \mu(h))$$

In a mixed strategy perfect recall game, the sequential equilibrium induces a positive probability on every node in a game tree in such that each player's strategy maximizes his expected utility in each information set, considering other players' strategies are fixed.

3.2.1 Solution Approach

The general approach of solving an extensive form game is to convert it to a normal form game by considering all pure strategies. But, the number of pure strategies of a large size extensive form game is often exponential. An alternative approach, called *sequence form*, is similar to normal form but linear in the game tree's size. The sequence form is a matrix where a sequence of sequential moves replaces pure strategies. Instead of moving from every information set, a player moves from a choice node by targeting a terminal node of the game tree in such that it creates a path from the root to the terminal node where each path goes through an information set of the player. This path represents a *sequence* rather than a pure strategy. A player's optimal strategy is to determine the probability distributions for his sequences to maximize his expected utility while other players have a fixed strategy. Therefore, the sequences can be characterized as variables to form the linear equations that can be solved using linear programming (LP). The solution of an LP results in the equilibria of a game.

3.3 Stackelberg Game

In the above-stated [Definition 1] normal form game, it is generally assumed that each player is symmetrically knowledgeable about the conditions of a game and he plays rationally to reach the Nash equilibria. This hypothesis is, however, limited to real-world scenarios. Stackelberg game, a popular non-symmetric game model, is used to analyze the interactions between a defender and an attacker when the defender keeps his strategy fixed. The game is proceeded by the defender making his move first as a leader who can play any mixed strategy. As a follower, the attacker evaluates the defender's strategy to devise an optimal strategy to respond to the defender. In this game, both leader and follower have a possible set of pure strategies denoted as δ and α, respectively. Each has a mixed strategy played over pure strategies using a probability distribution that is denoted as x for the leader and y for the follower.

Table 3.2: Example of a Stackelberg Game

Target	Defender		Attacker	
	Protected	Unprotected	Protected	Unprotected
1	0	-5	-5	10
2	-10	-20	-40	20

3.3.1 Stackelberg Equilibria

Stackelberg Equilibrium is the Stackelberg game's solution concept where the follower observes the leader's strategy x first then responds with strategy $f(x) : x \mapsto y$ which is optimal with respect to his expected payoff. Generally, Strong Stackelberg Equilibrium (SSE) and the Weak Stackelberg Equilibrium (WSE) are two types of Stackelberg equilibrium. In SSE, the follower breaks ties in favor of the defender, where in the WSE, he plays the worst strategy for the defender.

Definition 10 *(Strong Stackelberg Equilibrium [21, 22, 23]) A pair of strategies* $(x, f(x))$ *forms a Strong Stackelberg Equilibrium (SSE) if it satisfies the following:*

1. *The leader plays a best-response:* $U_l(x, f(x)) \geq U_l(x', f(x'))$*, for all the leader strategies* x'
2. *The follower plays a best-response:* $U_f(x, f(x)) \geq U_f(x, y')$*, for all the follower strategies* y'
3. *The follower breaks ties in favor of the leader* $U_f(x, f(x)) \geq U_f(x, y')$*, for all the follower strategies* y'

Chapter 4

Concealing Cyber-Decoys using Deception Games

4.1 Introduction

Both civilian and military computer networks are under increasing threat from cyber attacks, with the most significant threat posed by Advanced Persistent Threat (APT) actors. These attackers use sophisticated methods to compromise networks and remain inside, establishing greater control and staying for long periods to gather valuable data and intelligence. These attackers seek to remain undetected, and estimates from APT attacks show that they are often present in a network for months before they are detected [24].

Cyber deception methods use deceptive decoy objects like fake hosts (honeypots), network traffic, files, and even user accounts to counter attackers in a variety of ways [1, 2, 3]. They can create confusion for attackers, make them more hesitant and less effective in executing further attacks, and can help to gather information about the behavior and tools of various attackers. They can also increase the ability of defenders to detect malicious activity and actors in the network. This deception is especially critical in the case of APT attackers, who are often cautious and skilled at evading detection [25]. Widespread and effective use of honeypots and other deceptive objects is a promising approach for combating this class of attackers.

However, the effectiveness of honeypots and other deceptive objects depends crucially on whether the honeypot creators can design them to look similar enough to real objects, to prevent honeypot detection and avoidance. This design goal especially holds for APT

threats, which are likely to be aware of the use of such deception technologies and will actively seek to identify and avoid honeypots, and other deceptive objects, in their reconnaissance [25, 26]. A well-known problem with designing successful honeypots is that they often have characteristics that can be observed by an attacker that will reveal the deception [27]. Examples of such characteristics include the patterns of network traffic to a honeypot, the response times to queries, or the configuration of services which are not similar to real hosts in the network. However, with some additional effort, these characteristics can be made more effective in deception (e.g., by simulating more realistic traffic to and from honeypots).

We introduce a game-theoretic model of the problem of designing effective decoy objects that can fool even a sophisticated attacker. In our model, real and fake objects may naturally have different distributions of characteristic features than an attacker could use to tell them apart. However, the defender can make some (costly) modifications to *either* the real or the fake objects to make them harder to distinguish. This model captures some key aspects of cyber deception that are missing from other game-theoretic models. In particular, we focus on whether the defender can design convincing decoy objects, and what the limitations of deception are if some discriminating features of real and fake objects are not easily maskable.

We present several analyses of fundamental questions in cyber deception based on our model. We analyze how to measure the informativeness of the signals in our model, and then consider how effectively the defender can modify the features to improve the effectiveness of deception in various settings. We show how different variations in the costs of modifying the features can have a significant impact on the effects of deception. We also consider the differences between modifying only the features of deceptive objects, which can be referred to as one-sided deception or decoy deception only, and being able to modify both real and deceptive objects (two-sided deception). While this is not always necessary, in some cases, it is essential to enable effective deception. We also consider deception against naïve attackers, and how this compares to the case of sophisticated at-

tackers. Finally, we discuss how our model relates to work in adversarial learning and how this model could be applied beyond the case of honeypots to, for example, generating decoy network traffic.

4.2 Motivating Domain and Related Work

While the model we present may apply to many different types of deception and deceptive objects, we will focus on honeypots as a specific case to make our discussion more concrete and give an example of how this model captures essential features of real-world deception problems. Honeypots have had a considerable impact on cyber defense in the 30 years since they were first introduced [28].

Over time, honeypots have been used for many different purposes, and have evolved to more sophisticated designs with more advanced abilities to mimic real hosts and to capture useful information about attackers [29, 30, 31]. The sophistication of honeypots can vary dramatically, from limited low-interaction honeypots to sophisticated high-interaction honeypots [29, 32, 4].

Here, we do not focus on the technological advancements of honeypots, but rather on the game-theoretic investigation of honeypot deception. There have been numerous works that emphasize this game-theoretic approach to cyber deception as well. Our work builds upon the Honeypot Selection Game (HSG), described by Píbil et al. [33, 2]. Much like the HSG, we model the game using an extensive form game. We extend the HSG model with the introduction of *features*, which are modifiable tokens in each host that enable more robust deceptions and allow to model more realistic settings. Several game-theoretic models have been established for other cyber defense problems [34, 35, 36, 37], specifically for deception as well [38, 39], however these consider attribute obfuscation as the means of deception rather than use of decoy objects.

[40] notably investigates the use of honeypots in the smart grid to mitigate denial-of-service attacks through the lens of Bayesian games. [41] also model honeypots mitigating

denial-of-service attacks in a similar fashion but in the Internet-of-Things domain. [42] tackles a similar "honeypots to protect social networks against DDoS attacks" problem with Bayesian game modeling. These works demonstrate the broad domains where honeypots can aid. This work differs in that we do not model a Bayesian incomplete information game.

A couple of works also consider the notion of two-sided deception, where the defender deploys not only *real*-looking honeypots but also *fake*-looking real hosts. Rowe et al. demonstrate that using two-sided deception offers an improved defense by scaring off attackers [43]. Caroll and Grosu introduced the signaling deception game where signals bolster a deployed honeypot's deception [6]. Our work differs in that we define specific features (signals) that can be altered and revealed to the attacker. Shi et al. introduce the mimicry honeypot framework, which combines real nodes, honeypots, and *fake*-looking honeypots to derive equilibria strategies to bolster defenses [44]. They validated their work in a simulated network. This notion of two-sided deception is quickly becoming a reality; De Gaspari et al. provided a prototype proof-of-concept system where production systems also engaged in active deception [45].

4.3 Honeypot Feature Selection Game

We now present a formal model of the Honeypot Feature Selection Game (HFSG). This game models the optimal decisions for a player (the defender) who is trying to disguise the identity of real and fake objects so that the other player (the attacker) is not able to reliably distinguish between them. Each object in the game is associated with a vector of observable features (characteristics) that provides an informative signal that the attacker can use to detect fake objects more reliably. The defender can make (limited) changes to these observable features, at a cost. Unlike many models of deception, we consider the possibility that the defender can make changes to both the real and fake objects; we refer to this as 2-sided deception.

The original feature vector is modeled as a move by nature in a Bayesian game. Real and fake objects have different probabilities of generating every possible feature vector. How useful the features are to the attacker depends on how similar the distributions for generating the feature vectors are; very similar distributions have little information while very different distributions may precisely reveal which objects are real or fake. The defender can observe the features and may choose to pay some cost to modify a subset of the features. The attacker observes this modified set of feature vectors and chooses which object to attack. The attacker receives a positive payoff if he selects a real object, and a negative one if he selects a honeypot.

To keep the initial model simple, we focus on binary feature vectors to represent the signals. We will also assume that the defender can modify a maximum of one feature. Both of these can be generalized in a straightforward way, at the cost of a larger and more complex model.

4.3.1 Formal definition of Honeypot Feature Selection Game

We now define the Honeypot Feature Selection Game (HFSG) formally by the tuple $G = (K^{\mathrm{r}}, K^{\mathrm{h}}, N, v^{\mathrm{r}}, v^{\mathrm{h}}, C^{\mathrm{r}}, C^{\mathrm{h}}, P^{\mathrm{r}}, P^{\mathrm{h}}, \tau, \chi)$.

- K^{r} denotes the set of real hosts and K^{h} denotes the set of honeypots. Altogether, we have the complete set of hosts $K = K^{\mathrm{r}} \cup K^{\mathrm{h}}$. We denote the cardinalities of these by $k = |K|$, $r = |K^{\mathrm{r}}|$, $h = |K^{\mathrm{h}}|$.

- $[n]$ is the set of features that describe any given host. The sequence of feature values of a host is referred to as its *configuration*. Thus, the set of different possible configurations is $\{0, 1\}^n$.

- v^{r}, v^{h} denote the importance values of the real hosts and honeypots resp.

- C^{r}, C^{h} denote the cost vectors associated with modifying a single feature of a real host and a honeypot resp., and are indexed by the set of features N. Thus, C^{r}_i is the

cost of modifying the i^{th} feature of a real host.

- $P^{\mathrm{r}} : \{0,1\}^n \to [0,1]$ is probability distribution over feature vectors for real hosts
- $P^{\mathrm{h}} : \{0,1\}^n \to [0,1]$ is the probability distribution over feature vectors for honeypots
- The collection of all possible information sets is denoted by τ.
- $\chi : \{0,1\}^{kn} \times D \to \tau$ is a function that given the initial network and a defender action, outputs the attacker's resultant information set $I \in \tau$. Here, D is the set of defender actions.

An example of a small HFSG with 1 real host, 1 honeypot, and 1 feature for each host is shown in Figure 4.1. The probability distributions $P^{\mathrm{r}}(0) = P^{\mathrm{r}}(1) = 0.5$, and $P^{\mathrm{h}}(0) = P^{\mathrm{h}}(1) = 0.5$ are randomly generated for each feature combination.

4.3.2 Nature Player Actions

We assume that both players know the probability distributions P^r and P^h that define how the feature vectors are selected by nature for real and honeypot hosts, respectively. Nature generates the network configurations as per the distributions P^{r} and P^{h}. Thus, the network state $x = (x_1, \ldots, x_k)$ is generated as per the joint distribution P^{x} where $P^{\mathrm{x}}(x) = \prod_{i=1}^{r} P^{\mathrm{r}}(x_i) \times \prod_{i=r+1}^{k} P^{\mathrm{h}}(x_i)$. Both players can compute the distribution P^{x}. For example, in Figure 4.1 $P^{\mathrm{x}} = 0.25$ for network 0R1D is calculated from $P^{\mathrm{r}}(0) = 0.5$ and $P^{\mathrm{h}}(1) = 0.5$.

4.3.3 Defender Actions

The defender observes the network configuration $x \in X$, selected by nature as per probability distribution P^{x}. Then he chooses an appropriate action $d \in D$, which is to change at most one feature of any single host. Thus, D has $nk + 1$ different actions. This action

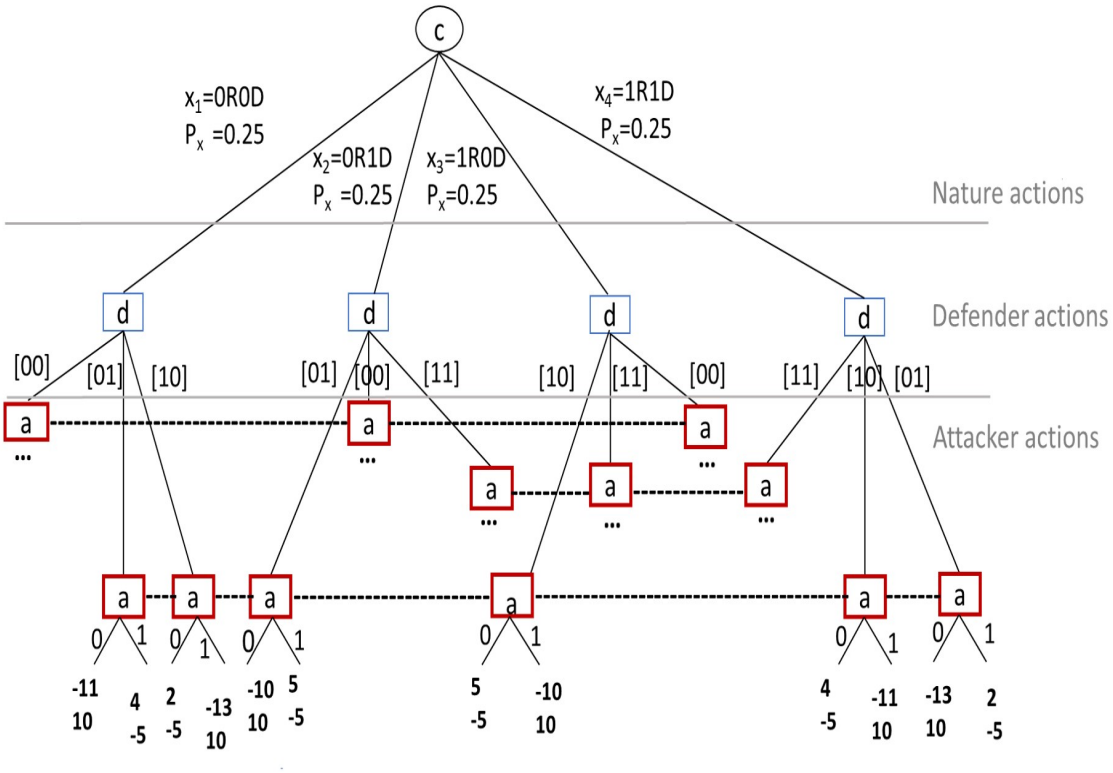


Figure 4.1: The extensive form game tree with one real host, one honeypot and 1 feature in each host. The importance value of real host is 10 whereas the modification cost of a feature is 3. The same values for the honeypot are 5, 1 resp.

results in a configuration $x' \in \{0,1\}^{nk}$ that the attacker observes, defining his information set $I \in \tau$ as described previously. In the example of Figure 4.1, given the initial network configuration 0R0D, the defender can alter a feature which results into 0R1D or 1R0D, or make no change leading to 0R0D as the attacker's observation.

4.3.4 Attacker Actions

The attacker observes the set of feature vectors for each network, but does not directly know which ones are real and which are honeypot. Thus, any permutation of the host configurations is perceived identically by the attacker. Hence, the attacker's information set is merely characterized by the combination of the host configurations and thus represented as a multiset on the set of host configurations as the Universe. For example, in Figure 4.1, the networks 0R1D and 1R0D belong to the same information set. Given the attacker's information set, he decides which host to attack. When indexing the attack options, we write the information set as an enumeration of the k host configurations, and we assume a lexicographically sorted order as a convention. Given this order, we use a binary variable a_i^I to indicate that when he is in the information set I, the attacker's action is to attack host $i \in K$.

4.3.5 Utility Functions

A terminal state t in the extensive form game tree is characterized by the sequence of actions that the players (nature, defender, attacker) take. The utilities of the players can be identified based on the terminal state that the game reaches. Thus, given a terminal state t as a tuple (x, j, a) of the player actions, we define a function $U(t) = U(x, j, a)$ such that the attacker gains this value while the defender loses as much. That is, this function serves as the zero-sum component of the player rewards. In particular, if the action a in the information set $\chi(x, j)$ corresponds to a real host, then $U(x, j, a) = v^{\mathrm{r}}$, whereas, if it corresponds to a honeypot, then $U(x, j, a) = -v^{\mathrm{h}}$. Intuitively, the successful identification of

a real host gives a positive reward to the attacker otherwise gives a negative reward that is equal to the importance value of a honeypot. The expected rewards are computed by summing over the terminal states and considering the probabilities of reaching them. Finally, the defender additionally also incurs the feature modification cost C_i^{r} or C_j^{h} if his action involved modifying i^{th} feature of a real host or j^{th} feature of a honeypot respectively.

4.3.6 Defender's Linear Program

We can solve this extensive form game with imperfect information using a linear program. For solving this game in sequence form [46], we create a path from the root node to the terminal node that is a valid sequence and consists of a list of actions for all players. Then we compute defender's behavioral strategies on all valid sequences using a formulated LP as follows, where U_d and U_a are the utilities of the defender and the attacker. To solve the program, we construct a matrix $X[0 : 2^{kn}]$ of all possible network configurations, and then the defender chooses a network $x \in X$ to modify. In network x, any action d of the defender leads to an information set I for the attacker. Different defender's actions in different networks can lead to the same information set $I \in \tau$. Then, in every information set I, the attacker chooses a best response action to maximize his expected utility.

$$max \sum_{x \epsilon X} \sum_{j \epsilon D} \sum_{i \epsilon K} U_d(x,j,i)\ d_j^x P^x a_i^{\chi(x,j)} \tag{4.1}$$

$$s.t. \sum_{(x,j):\chi(x,j)=I} U_a(x,j,i) d_j^x P^x\ a_i^I \geq \sum_{(x,j):\chi(x,j)=I} U_a(x,j,i')\ d_j^x P^x\ a_i^I$$

$$\forall i, i' \in K \quad \forall I \in \tau \tag{4.2}$$

$$d_j^x \geq 0 \quad \forall x \in X \quad \forall j \in D \tag{4.3}$$

$$\sum_{j \epsilon D} d_j^x = 1 \quad \forall x \in X \tag{4.4}$$

$$\sum_{i \epsilon K} a_i^I = 1 \quad \forall I \in \tau \tag{4.5}$$

The program's objective is to maximize the defender's expected utility, assuming that the attacker will also play a best response. In the above program, the only unknown variables are the defender's actions D (the strategies of a defender in a network $x \in X$) and the attacker's actions a^I. The inequality in Equation 7.4 ensures that the attacker plays his best response in this game, setting the binary variable a_i^I to 1 only for the best response i in each information set. Equation 4.3 ensures that the defender strategies in a network x is a valid probability distribution. Equation 4.4 makes sure that all probability for all network configurations sum to 1. Finally, Equation 4.5 ensures that the attacker plays pure strategies.

4.4 Empirical Study of HFSG

The HFSG game model allows us to study the strategic aspects of cyber deception against a sophisticated adversary who may be able to detect the deception using additional observations and analysis. In particular, we can evaluate the effectiveness of cyber deception under several different realistic assumptions about the costs and benefits of deception, as

well as the abilities of the players. We identify cases where deception is highly beneficial, as well as some cases where deception has limited or no value. We also show that in some cases, using two-sided deception is critical to the effectiveness of deception methods.

4.4.1 Measuring the Similarity of Features

One of the key components of our model is that real hosts and honeypots generate observable features according to different probability distributions. The similarity of these distributions has a large effect on the strategies in the game, and the outcome of the game. Intuitively, if out-of-the-box honeypot solutions look indistinguishable from existing nodes on the network the deception will be effective without any additional intervention by the defender. However, when the distributions of features are very dissimilar the defender should pay higher costs to modify the features to disguise the honeypots. In some cases this may not be possible, and the attacker will always be able to distinguish the real hosts and honeypots.

Measuring the similarity of the feature distributions is a somewhat subtle issue, since the defender can make changes to a limited number of features. Standard approaches such as Manhattan distance or Euclidean distance do not provide a good way to compare the similarity due to these constraints. We use a measure based on the Earth Mover's Distance (EMD) [47], which can be seen as the minimum distance required to shift one pile of earth (probability distribution) to look like another. This measure can be constrained by the legal moves, so probability is only shifted between configurations that are reachable by the defender's ability to change features.

In the experiments, we allow the defender to modify only a single feature in the network and the EMD determines the minimum cost needed to transform a weighted set of features to another where the probability of each feature configuration is the weight. The ground dissimilarity between two distributions is calculated by the Hamming distance. This distance between two distributions of equal length is the number of positions at which

the comparing features are dissimilar. In other words, it measures the minimum number of feature modification or unit change required to make two sets of feature indistinguishable. We model the distance from moving the probability of one configuration (e.g., turning $[0, 0]$ into $[0, 1]$) to another by flipping of a single bit at a time with a unit cost of 1. This can be seen visually in Figure 4.2 where we calculate the EMD of moving the honeypot's initial distribution into that of the real node's initial distribution.

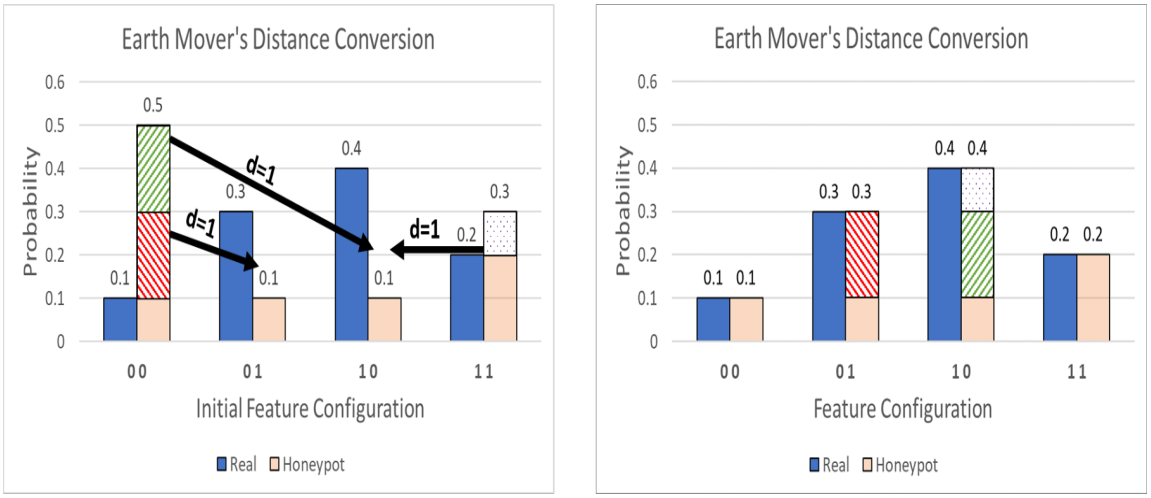
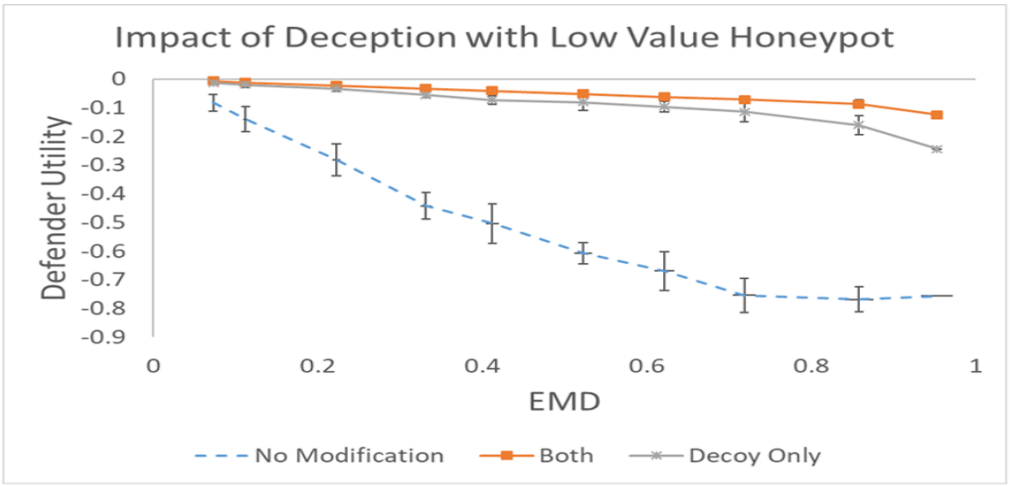


Figure 4.2: Earth Mover's Distance process. a) Displays the initial feature configuration probability distributions P_r and P_h and where to move slices of the distribution from P_h and b) Shows the updated P_h after the conversion, resulting in a final EMD of 0.5.

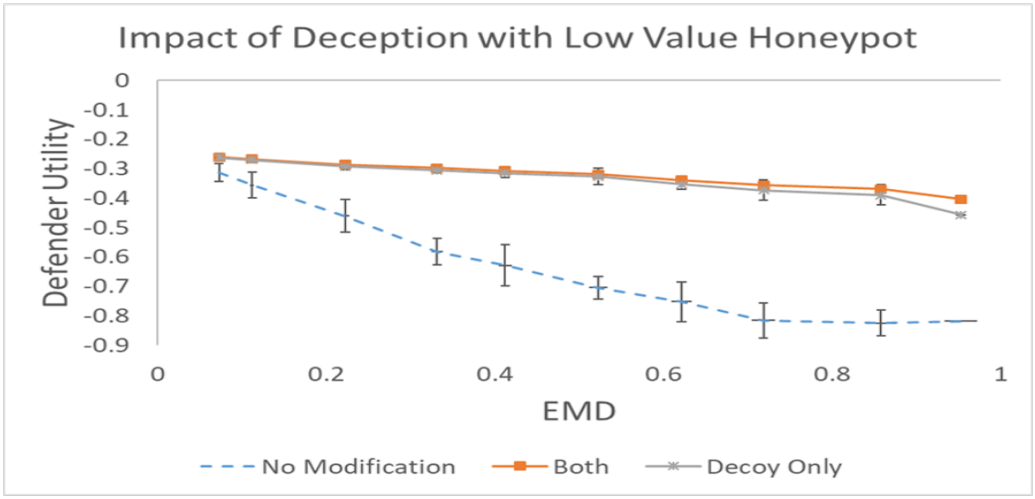
In our experiments we will often show the impact of varying levels of similarity in the feature distributions. We generated 1000 different initial distributions for the features using uniform random sampling. We then calculated the similarities using the constrained EMD and selected 100 distributions so that we have 10 distributions in each similarity interval. We randomly select these 10 for each interval from the ones that meet this similarity constraint in the original sample. This is necessary to balance the sample because random sampling produces many more distributions that are very similar than distributions that are further apart, and we need to ensure a sufficient sample size for different levels

of similarity. we present the results by aggregating over the similarity intervals of 0.1 and average ten results in each interval.

4.4.2 Deception with Symmetric Costs



(a)



(b)

Figure 4.3: Comparison of defender utility when the real host's importance value a) doubles that of the honeypot and b) equals that of the honeypot. Here we see one-sided deception provides a comparable defense despite a high initial dissimilarity.

Table 4.1: Parameters used in HFSG experiments. RIV denotes real system's importance value, RMC denotes real system's feature modification cost, HpIV denotes importance value of honeypot and HpMC denotes feature modification cost of honeypot. All numbers are normalized to 1

Figure	RIV	RMC		HpMC		HpIV
		F 1	F 2	F 1	F 2	
4.3a	1.0	0.25	0.1	0.1	0.25	0.5
4.3b	1.0	0.25	0.1	0.2	0.1	1.0
4.4 (Both (A))	1.0	0.25	0.1	0.1	0.2	0.5
4.4 (Both (B))	1.0	0.5	0.2	0.1	0.2	0.5
4.4 (Both (C))	1.0	1.0	0.5	0.1	0.2	0.5
4.5 (Exp-1)	1.0	0.1	∞	0.1	∞	1.0
4.5 (Exp-2)	1.0	0.1	∞	∞	0.1	1.0
4.6 (Exp-1)	1.0	0.2	0.2	0.2	0.2	1.0
4.6 (Exp-2)	1.0	0.15	0.25	0.25	0.15	1.0
4.6 (Exp-3)	1.0	0.1	0.3	0.3	0.1	1.0
4.6 (Exp-4)	1.0	0.05	0.35	0.35	0.05	1.0
4.6 (Exp-5)	1.0	0.0	0.4	0.4	0.0	1.0
4.8	1.0	0.25	0.1	0.2	0.1	1.0

Our first experiment investigates the impact of varying the similarity of the feature distributions. We also vary the values of real host and honeypot. As the similarity of the distributions P^r and P^h decreases, we would expect a decrease in overall expected defender utility. We can see this decrease in Figures 4.3a and 4.3b as we vary the similarity measured using EMD. In Figures 4.3a and 4.3b, we compare the utility differences between an optimal defender that can only modify the features of the honeypot (one-sided deception), an optimal defender that can modify features of *both* the honeypot and real host (two-sided deception), and a baseline defender that cannot make any modifications against a fully rational best response attacker.

In Figure 4.3a, the honeypot has the same importance value as the real host, while in Figure 4.3b, the honeypot value is half of the real host. The first observation is that in both cases the value of deception is high relative to the baseline with no deception, and this value grows dramatically as the feature distributions become more informative (higher EMD). In general, the defender does worse in cases where the hosts have different values. Two-sided deception does have a small advantage in cases with highly informative features, but the effect is small. Here, the costs of modifying the features are symmetric, so there is little advantage in being able to modify the feature on either the honeypot or the real host, since the defender can choose between these options without any penalty.

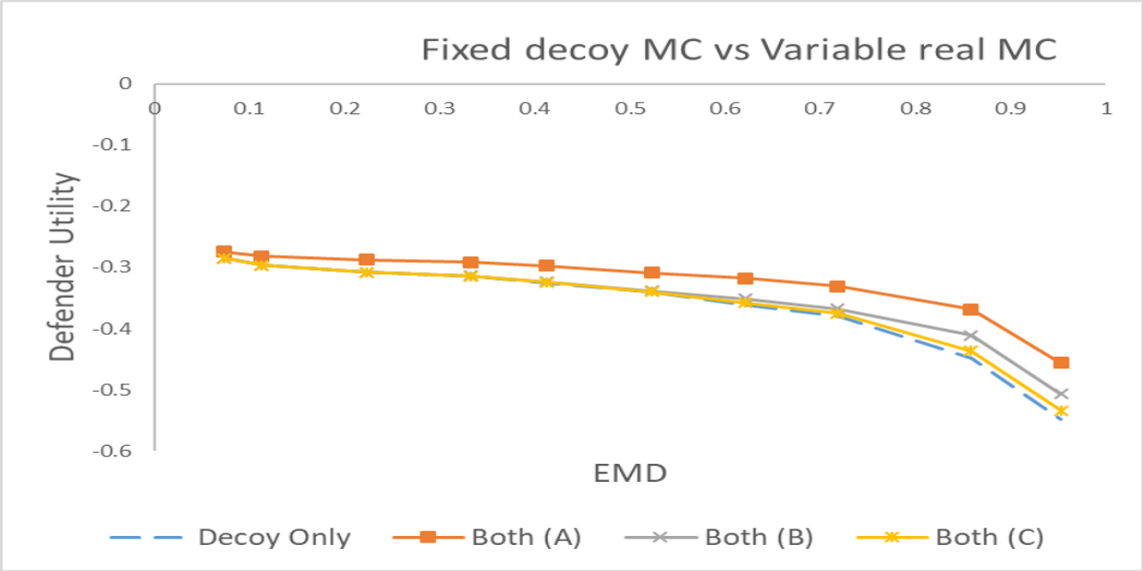


Figure 4.4: Comparison of defender utility when the cost of modifying the real host features is different than modifying the honeypot features.

To further investigate the issue of one-sided and two-sided deception, we fix the honeypot features modification costs and increased real host modification costs as reflected in Table 4.1. Here, we compare how increasing the real host's feature modification negatively affects the defender's expected utility. As the cost for modifying the real hosts increases relative to the cost of modifying honeypots, the defender must make more changes on honeypots in order to maximize his utility. Altering the real system in this case is not feasible and does not provide a good return on investment.

Traditionally network administrators avoid altering features in their real hosts on the network and simply employ one-sided deception, attempting to alter the honeypot to look like a real host. In the case where modifying a real host to look *less believable* might be be too costly or even impossible, one-sided deception is an obvious choice as demonstrated in Figure 4.4. However, when these real feature modifications are not too costly, we see that two-sided provides a noticeable increase in defenses when the feature distributions are increasingly dissimilar.

4.4.3 Deception with Asymmetric Costs

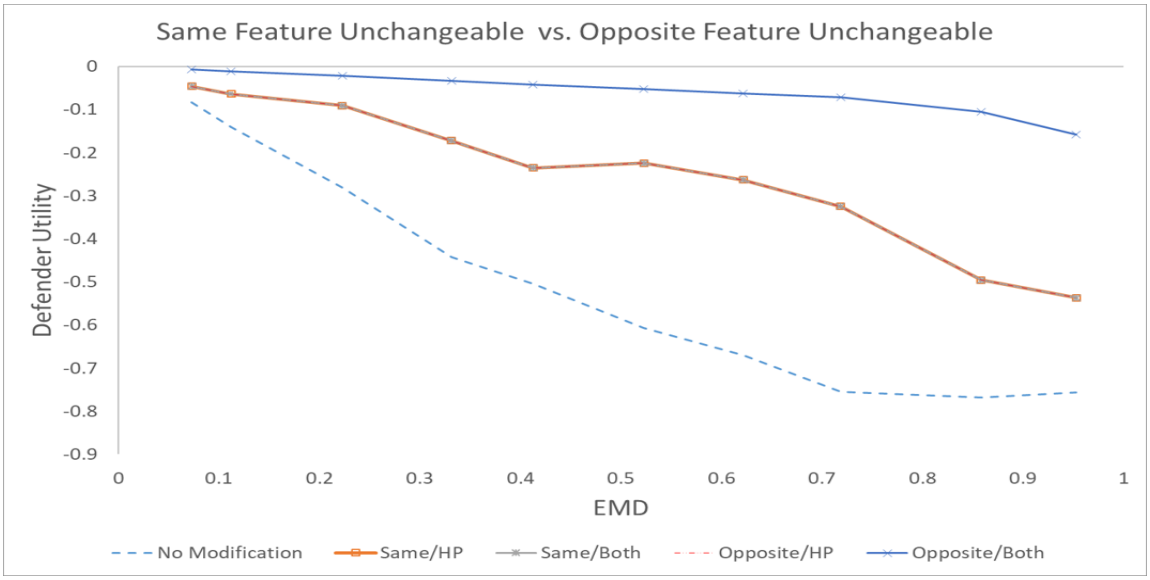


Figure 4.5: Comparison of defender utility when some features cannot be modified.

While the results so far have suggested that one-sided deception may be nearly as effective as two-sided deception, they have all focused on settings where the costs of modifying features are *symmetric* for real and fake hosts. We now investigate what happens when the costs of modifying different features are asymmetric. We start with the extreme case where some features may not be possible to modify at all.

In our examples with two features, we can set the unmodifiable features for the real and honeypot hosts to be the same or to be opposite. In Figure 4.5, we show the results of the game when we set the modification costs of some features to infinity. If the same feature for the real host and honeypot are unmodifiable, then there is little the defender can do to deceive an intelligent attacker when they are highly dissimilar. However, when the features that cannot be modified are different for the real and honeypot hosts, we see a very different situation. In this case the defender benefits greatly from being able to use two-sided deception, since he can avoid the constraints by modifying either the real or fake hosts as needed.

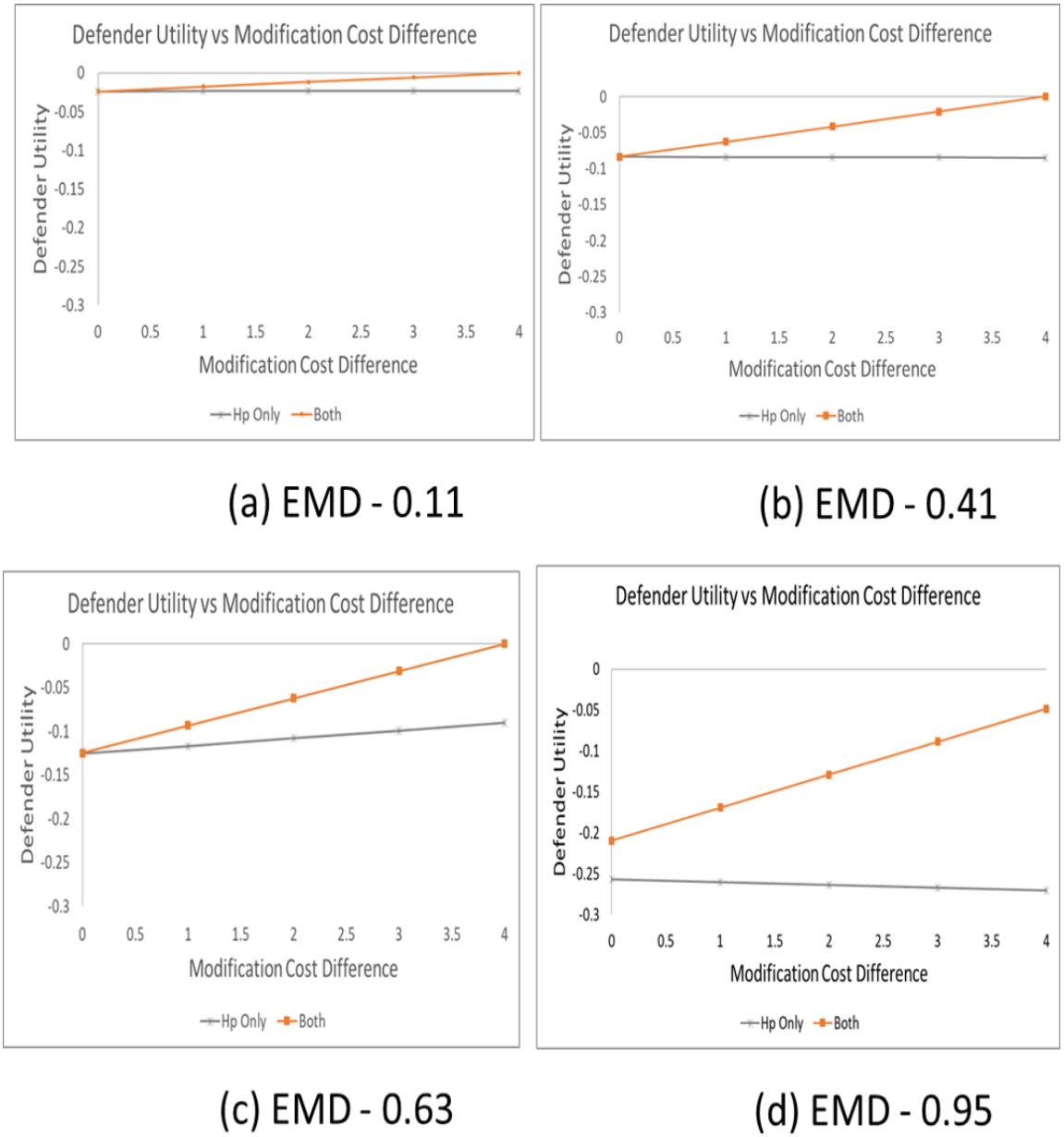


Figure 4.6: Impact of modification cost over various initial similarity parameters.

In our next experiment, we investigate less extreme differences in the costs of modifying features. We set the costs so that they are increasingly different for real and honeypot hosts, so modifying one feature is cheap for one but expensive for the other, but not impossible. We show the results of using either one or two-sided deception for varying levels of

initial feature distribution similarity in Figure 4.6. The specific costs are given in Table 4.1. We see that there is very little difference when the initial distributions are similar; this is intuitive since the attacker has little information and deception is not very valuable in these cases. However, we see a large difference when the initial distributions are informative. As the difference in the feature modification costs increases, the value of two-sided deception increases, indicating that this asymmetry is crucial to understanding when two-sided deception is necessary to employ effective deception tactics.

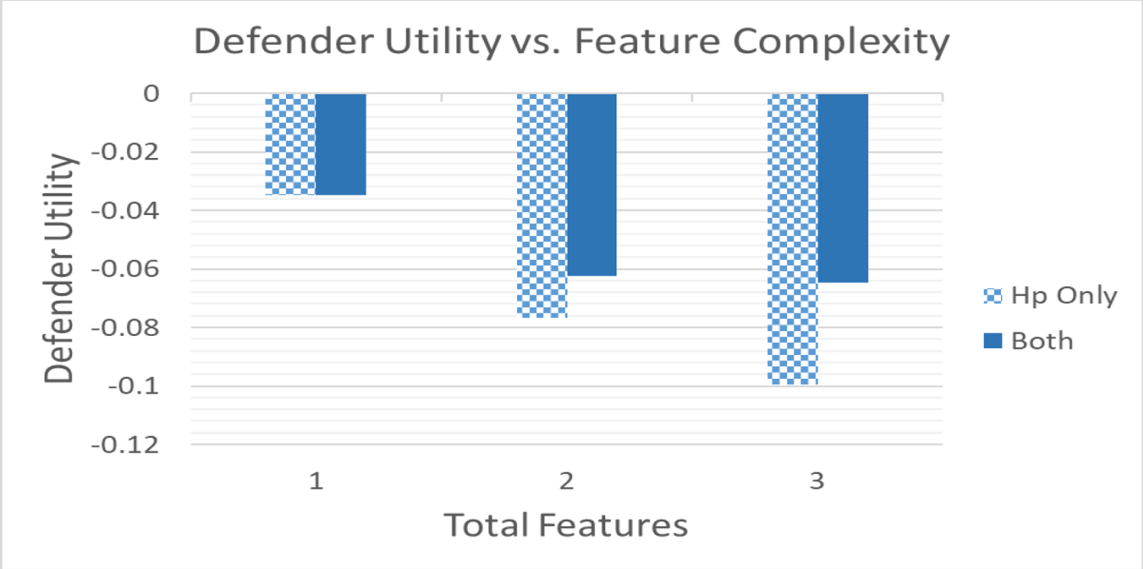


Figure 4.7: Comparison of defender utility when increasing the number of features.

We also expect that the number of features available to the players will have a significant impact on the value of deception. While the current optimal solution algorithm does not scale well, we can evaluate the differences between small numbers of features, holding all else equal. Figure 4.7. presents the results of the modeling HFSG with variable number of features.We found that when the number of features is increased two-sided deception becomes more effective than one-sided deception. The defender in this case has more opportunity to alter the network by changing the features and make it the more confusing network to the attacker. However, the defender payoff decreases with more features due to the constraint on how many features he can modify and the total cost of modifying these

features.

4.4.4 Deception with Naïve Attackers

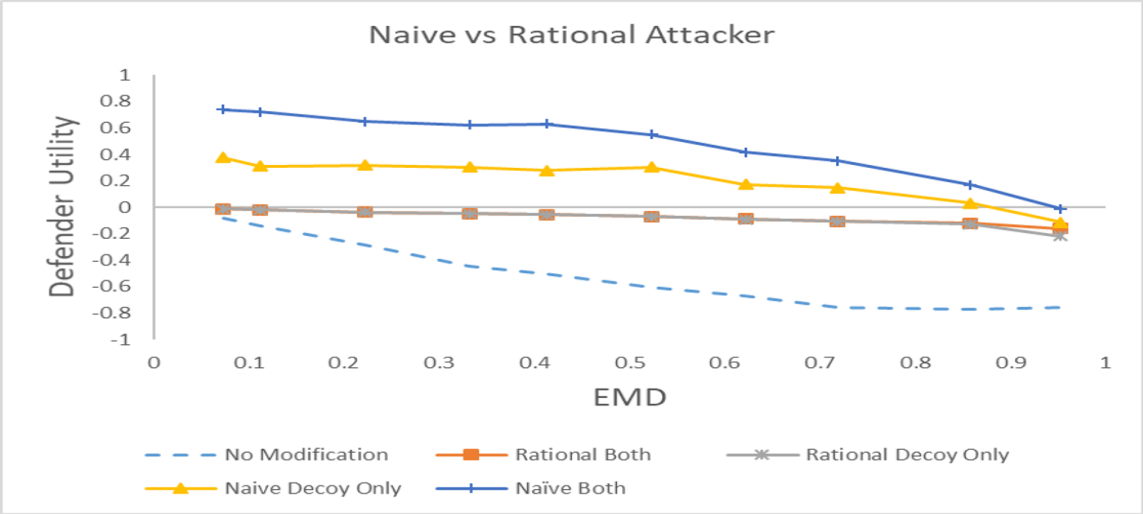


Figure 4.8: Comparison of defender utility of a naïve attacker versus a fully rational attacker. Here, the naïve attacker does not consider the defender's utility or strategy at all.

The previous empirical results all assumed a cautiously rational attacker who actively avoided attacking honeypots. This is a common practice, because fully rational actors present the highest threat. In cybersecurity, these fully rational attackers might be an experienced hacker or APT. However, these are not the only threats faced in cybersecurity and we cannot assume that these attacking agents are always cautious and stealthy. For example, many attacks on networks may be conducted by worms or automated scripts that are much simpler and may be much more easily fooled by deceptive strategies.

We now consider a more naïve attacker that does not consider the defender's deception. He observes the hosts on the network and assumes no modifications were made. Based on all observations for a particular network he calculates his best response, but does predict the defender's optimal strategy. The results of the experiment are shown in Figure 4.8 and the costs given in Table 4.1.

The best case is when the defender can perform two-sided deception against a naïve attacker and the worst case is when the defender performs no deceptive actions against a fully rational attacker. These two cases form an upper- and lower-bound as seen in Figure 4.8. Two-sided deception is more effective in this case when the feature distributions are similar, while the opposite was true for a rational attacker. Overall, deception strategies are much more effective against naïve attackers.

4.5 Discussion and Further Applications

Our model gives a new and more nuanced way to think about the quality of different deception strategies, and how robust they are to an adversary being able to see through the deception. We can identify which features the defender should focus on modifying to make the deception more effective, including features of the real objects. In addition, we can correctly identify cases where deception is not the best solution because the costs of creating a believable deception may be higher than the value they create. One of our key findings is that the two-sided deception is an effective strategy when the situation is asymmetric. One practical example is that activating on or off a particular process of a host is more expensive than enabling or disabling a specific port. Moreover, running an extra process on a real host may result in higher overhead than a honeypot. Therefore, when the costs of feature modification differ, the modification space increase, resulting in an effective deception practice. Finally, we conclude by discussing some connections to adversarial machine learning and an additional case where our model could be applied beyond honeypots.

4.5.1 Adversarial Learning

Recently, adversarial machine learning models have shown great promise in generating deceptive objects, focusing mostly on images and video applications [48, 49, 50], though they have the potential to generalize to many other types of deceptive objects. The most

well-known approach is Generative Adversarial Networks (GAN) [51], which rely on a pair of neural networks, one to generate deceptive inputs, and the other to detect differences between real and fake inputs. The intuition for these is often that the networks are playing a zero-sum game, though the interpretation is vague and there is no formal game presented. Our model can be viewed as a formalization of the game these types of machine learning algorithms are playing, though there are some differences. We specifically consider the costs of modifying different features of the objects, as well as the possibility of modifying the real distribution in addition to the fake one. On the other hand, GANs typically are used in much larger problems with vast numbers of complex features, and they do not find optimal solutions. Also, they use abstracted representations of the feature space in the learning process, and it is not clear exactly how this works or what the implications are.

We believe that further developing and scaling this model to address more complex feature deception problems will help to understand the theoretical qualities of GANs and related methods better. In particular, we can better understand the limits that these AML methods may have based on the costs and infeasibility of modifying features in some cases, as well as giving optimal or bounded approximations of the solutions to small feature deception games, which can then be used to provide clear quality comparisons for machine learning methods that may scale to much more complex problems but without specific quality guarantees.

4.5.2 Disguising Network Traffic

While we presented our model using honeypots as a motivating domain, there are many other possible applications. We briefly discuss another example here to make this point. There are many reasons to disguise network traffic to look like other traffic; defenders may wish to do this to generate fake traffic to support honeypots or to conceal the properties of real traffic on their networks otherwise. Attackers also may want to make their network

traffic appear similar to real traffic to avoid detection.

While network traffic, in general, has a very large number of possible features, an increasing fraction of traffic is encrypted, which hides many of the deep features of the data. However, it is still possible to do an analysis of encrypted traffic based on the source, destination, routing, timing and quantity characteristics, etc. Our model can be used to analyze how to optimize the properties of real and decoy traffic to improve the effectiveness of the decoy traffic, based on the costs of modifying different features. For example, modifying traffic to be sent more frequently will clearly have costs in increased network congestion, while modifying some features of the real traffic may not be feasible at all (e.g., the source and destination). Even simple versions of our model with relatively few features could be used to optimize decoy network traffic in encrypted settings, where there is limited observable information about the traffic. The unencrypted case allows for many more possible features, so it would require larger and more complex versions of our model to analyze, which would require more scalable algorithms to solve exactly using our model, or the application of approximation methods and adversarial learning techniques.

4.5.3 Limitations

The time and memory complexities of the game model depend on n, k, feature modification options, and the amount of sampling; which makes the model grow exponentially. To avoid computational complexity, we tested our model with two machines, one each of type (*real* and *honeypot*) with two features in each. Extending the model to include more machine types and features is straightforward, although the optimization problem will become much more difficult to solve. A scalable algorithm will need to be developed to solve larger size games.

4.6 Conclusions

Deception is becoming an increasingly crucial tool for both attackers and defenders in cybersecurity domains. However, existing formal models provide little guidance on the effectiveness of deception, the amount of effort needed to sufficiently disguise deceptive objects against motivated attackers, of the limits of deception based on the costs of modifying the features of the deceptive objects. Also, most analyses only consider how to make deceptive objects look real, and not how real objects can be modified to look more like deceptive ones to make the task of deception easier. We present a formal game-theoretic model of this problem, capturing the key problem of disguising deceptive objects among real objects when an attacker may observe external features/characteristics.

Our model of HFSG allows us to investigate many aspects of how a defender should optimize efforts to conceal deceptive objects, which can be applied to honeypots, disguising network traffic, and other domains. This also gives a more theoretical foundation to understand the benefits and limitations of adversarial learning methods for generating deceptive objects. We show that the symmetry or asymmetry of the costs of modifying features is critical to whether we need to consider two-sided deception as part of the strategy, and we also show that in some cases deception is either unnecessary or too costly to be effective. Also, the sophistication of the attackers makes a great difference; in cases with naïve attackers deception is even more effective, even when considering a low-cost strategy.

Chapter 5

Two-Sided Feature Deception Using Adversarial Learning

5.1 INTRODUCTION

Two-sided deception is a technique of making fake objects look real, and in many cases, making real objects look fake is a better form of deception. This concept of two-sided deception is rapidly gaining attention; De Gaspari et al. created a prototype proof-of-concept system in which production systems engaged in active deception [45]. When a defender uses real-looking honeypots and fake-looking real hosts, it provides better defense by scaring away attackers [43]. Shi et al. offer the mimicry honeypot framework, which mixes real nodes, honeypots, and fake-looking honeypots to enhance defenses where the findings are simulated in a network [44]. Aggarwal et al. [52]conducted an exploratory study based on human attackers, which found that human attackers are confused more when masking feature characteristics of real and fake objects rather than using one-sided deception. Miah et al. [53] proposed a Honeypot Feature Selection Game (HFSG) model in which a defender can modify some features of either the real or decoy objects (at some cost), making the decoys more effective. The game model determines optimal strategies for generating fake and real objects, but the algorithm is not scalable for a large size game model.

This work builds upon Honeypot Feature Selection Game (HFSG), described by Miah et al. [53]. One limitation of the current HFSG model is that the two-sided deception approach is applied on a small number of features. But we can connect the model to

adversarial machine learning methods where two-sided deception could be applied to a large set of features. It will help us find how to extend and scale the two-sided deception approach to understand adversarial learning's theoretical qualities. Recently, adversarial machine learning models have shown great promise in generating deceptive objects and the most well-known approach is Generative Adversarial Networks (GAN) [51, 54]. Typically GANs are used in much larger problems with vast numbers of complex features and generate fake samples without modifying real samples. On the other hand, adversarial attacks try to fool machine learning classifiers by changing original samples. The attackers can generate adversarial samples by adding small perturbation to the original inputs intent to mislead machine learning models. [55, 56]. They can also train their own models with adversarial samples and transfer the samples to the victim model in order to produce incorrect output by the victim classifier. [57]. Currently, no method is fully effective against adversarial examples. [58, 59]. We combine adversarial attack and GAN models to analyze two-sided deception approach on a large scale.

In this work, we propose using Adversarial Machine Learning (AML) methods to solve the problem of two-sided feature deception. We model the problem as a two-player minimax game between a defender and an attacker. The defender attempts to confuse the attacker by creating fake and adversarial samples, the latter of which is created by modifying a real sample in such a way that the attacker mispredict it as a fake sample. The attacker's goal is to correctly distinguish between real and fake samples to minimize his expected loss. On the other hand, the defender's strategy is to maximize the attacker's expected loss. Our contributions are three-fold:

- We introduce the two-sided Generative Adversarial Network (TS-GAN), a novel algorithm for capturing two-sided feature deception approach. Though, the algorithm is limited to provide optimal solutions, but provide approximation solution for two-sided feature deception problem.

- Our method can generate fake samples look likes as real samples and real samples

looks like as fake samples when the feature space is complex and large. In a dynamic learning environment, the technique can also be used as a robust classifier for the binary classification problem.

- We present an empirical analysis of this model to show strategies for effectively generating adversarial examples that may be used for adversarial attacks successfully in a semi-white box setting.

5.2 Background

5.2.1 Generative Adversarial Networks

Generative Adversarial Networks (GAN) is one of the most successful models for unsupervised generative modeling where two players compete with each other: the generator (G) tries to trick the Discriminator (D) into classifying its generated fake data as true [54, 51]. It is a probabilistic model that uses noise variables z and observable variables x for learning. The model corresponds to a minimax zero-sum game where two functions can represent the two players in the game. Generally, in practice, both are implemented using multilayer networks. To produce fake samples from the same distribution as the real data x, the prior input noise variables, $p_z(z)$, are created to learn the generator's data distribution p_g over x. The G generates a data distribution using $G(z;\theta_g)$, where G is a differentiable function define by a multilayer network with θ_g parameters. Similarly, D is also a multilayer network can be defined by $D(x;\theta_d)$ where the θ_d are the model parameters. The probability that x came from the data rather than p_g is defined by $D(x)$. In this model, D is trained to maximize the probability of assigning the correct label to both training examples and samples from G. Simultaneously, G is trained to minimize $\log(1 - D(G(z)))$. Therefore, the value function $V(G;D)$ of the minimax zero-sum game can be defined as follows :

$$\min_G \max_D V(G,D) = \mathbb{E}_{x\sim P_{data}(x)}\left[\log D(x)\right] + \mathbb{E}_{z\sim P_z(z)}\left[\log(1 - D(G(z)))\right] \qquad (5.1)$$

D and G compute their gradients simultaneously and adversarially in the above game, each making one step in the gradient's direction and repeating until neither can make progress. Though GAN is theoretically zero-sum, the learning processes of D and G are different. The global optimum of the minimax game model is reached when $p_g = p_{real}$, but this might be computationally infeasible to achieve.

5.2.2 Adversarial Examples

Machine learning algorithms are frequently vulnerable to adversarial attacks. An adversary can carefully add small perturbations to the original input data that can mislead the machine learning models with high probability. He can even train his model with adversarial samples and then transfer the samples to the victim model, causing the victim classifier to provide inaccurate output. One popular technique to generate an adversarial example with a given classifier is to alter the input in a way that increases the cross-entropy loss. If an input x that is correctly classified by neural network $h(x)$ and $h^*(x)$ is the true class. The adversarial learning algorithm then looks for a small perturbation to x, $x^{adv} = x + \Delta x$, such that $h^*(x) \neq h(x^{adv})$. The value of Δx should be small enough when added to x and the difference between x^{adv} and x should be almost imperceptible. The objective of the learning algorithm is to minimize the probability assigned to the true label. For the neural network h, network parameters θ, input data x, and corresponding true output t^{true}, the adversary determines the perturbation Δx as follows:

$$\Delta x = arg \max_{r \sim \rho} J(\theta, x + r, t^{true}) \tag{5.2}$$

Equation (5.2) is a optimization problem that maximizes the objective function J where ρ is the adversary's policy, a deterministic mapping from input x to the space of bounded perturbations $r \leq \lambda$.

While there are a many methods for adversarial examples, the L-BFGS algorithm [50] generates an adversarial example x^{adv} from a input x is as follows :

$$min||x - x^{adv}||_2 + \lambda J(h_\theta(x^{adv}), t^{true}) \tag{5.3}$$

The first term imposes a penalty of x for large perturbations, whereas the second imposes a penalty when classification deviates from the target class t^{true}. J denotes the loss function between t^{true} and the output of the classifier $h_\theta(x^{adv})$. The model parameter is $\lambda > 0$.

For, a given input x, label and a fixed model, the fast gradient sign method [55] generates untargeted adversarial perturbations by performing one step in the gradient sign's direction with step-width ϵ as follows :

$$x^{adv} = x + \epsilon \; sign(\Delta_x J(h_\theta(x^{adv}), t^{true})) \tag{5.4}$$

This method determines an adversarial perturbation under L_∞ norm where the norm computes the maximum change to any of the coordinates.

5.3 Two-Sided Generative Adversarial Network

We can consider the two-sided adversarial learning problem from a game-theoretic perspective, the defender aims to generate fake and adversarial examples in order to minimize its maximum possible loss against the best policy of the adversary. We modify the GAN's formulation to an adversarial training scenario in which the defender and adversary compete in minimax game, with the attacker acting as the discriminator. We consider that the two-sided Generative Adversarial Network (TS-GAN) consists of two parts, the attacker, and the defender. The defender learns to create perturbation for a given input data as well to generate fake data from a latent space. The attacker learns to be robust to the, real, perturbed and fake data. Figure(5.1) shows the basic architecture of TS-GAN.

Now, we formulate our problems as follows: Let's $(x_1, ..., x_n)$ represent the training instances, and P_{real} represent the uniform distribution across $(x_1, ..., x_n)$. Here, (x_i, y_i) is the i^{th} instance in the training set, which is made up of feature vectors $x_i \in \chi$ where $\chi \subseteq \mathbb{R}^n$ represents the feature space and y_i corresponding real class label (1). Also, let $G(z_1), ..., G(z_r)$ be a collection of r examples from the generated distribution Pg that are corresponding fake class label (0) and represents by $(x_1^{'}, ..., x_n^{'})$ where $x_i^{'} \in \chi$. Similarly,

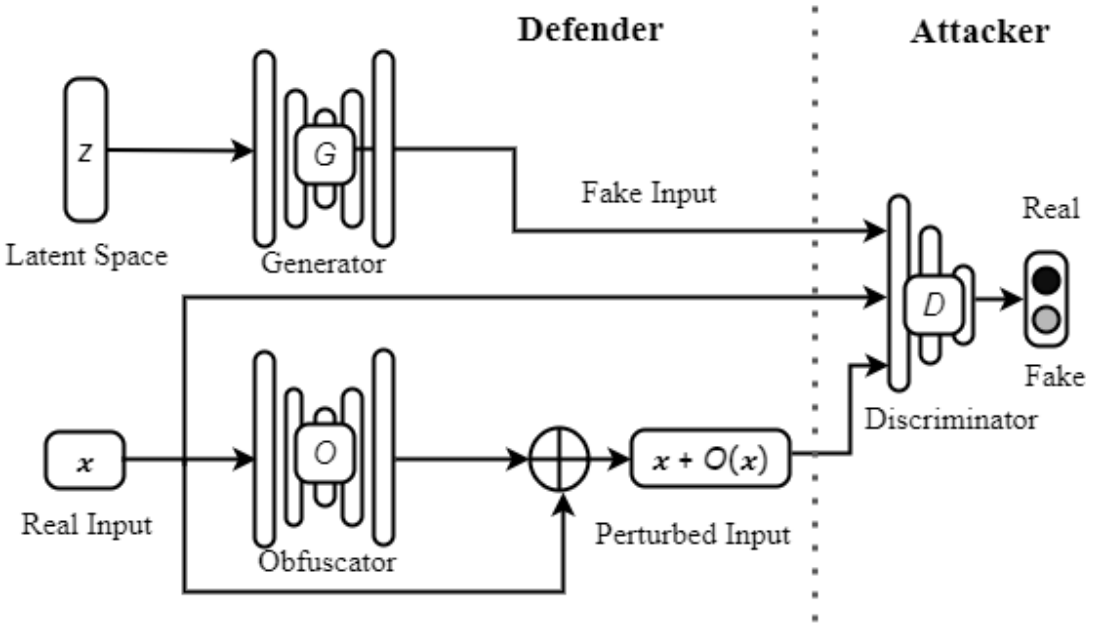


Figure 5.1: Two-Sided Generative Adversarial Network Architecture

assume, $O(x_1), ..., O(x_n)$ is a set of perturbation generated from $(x_1, ..., x_n)$ where $x_i + O(x_i) = x_i^{adv} \in \chi$ is the i^th adversarial example, such that $D(x_i^{adv}) = t$ (target attack) where t is the target class (0). The adversary's learning goal is to learn a classifier $D : \chi \to \mathbb{Y}$ from the domain χ to the set of classification outputs $\mathbb{Y} \in \{0, 1\}$, where $|\mathbb{Y}|$ represents the number of classification outputs. Now, we formally define TS-GAN, which consists of a defender and an attacker like following.

5.3.1 Defender

The defender, we assume, contains two modules, both of which would be neural networks. One of the networks is the generator that generates fake data which we represent as G_θ with θ parameters. The G_θ uses latent space z from a l-dimensional spherical Gaussian distribution P_g, then applies G_θ to h to create a fake sample $x' = G_\theta(z)$ of the distribution P_g. Also, it learns to estimate the distribution from which the training data is drawn to

generate fake samples from that distribution P_g. The objective of the G_θ is to minimize the probability of generated sample as fake by the attacker. Therefore, the loss functions of the G_θ is specified as:

$$\mathcal{L}_\mathcal{G} = -\mathbb{E}_{z\sim P_z(z)}\left[\phi(1 - D_\theta(G_\theta(z)))\right]$$

The defender's second neural network is Obfuscator, which refers to O_θ with θ parameters. The O_θ takes the original instance x as its input and generates a perturbation $\Delta_x = O_\theta(x)$. The dimensions of the inputted data and outputted perturbed data of this network are identical. Then $x + O_\theta(x)$ will be passed to the discriminator D_θ, which will learn the samples from the real sample distribution using the obfuscated data. The goal of O_θ is to make it so that a given adversarial example and generated fake samples are indistinguishable by D_θ. To fulfill the goal of fooling D_θ's learning, we perform an iterative adversarial attack where the target model D_θ takes $x + O_\theta(x)$ as its input and outputs its loss $\mathcal{L}^D_{adv}$, which represents the distance between the prediction and the target class t (targeted attack). Here, the discriminator D_θ aims to distinguish the adversarial samples $x^{adv} = x + O_\theta(x)$ from the fake data $x^{'}$. Note that the $x^{'}$ is sampled from the fake class, so as to encourage that the generated adversarial examples are close to the fake class. Therefore, the loss for fooling the D_θ in a targeted attack is:

$$\mathcal{L}^D_{adv} = \mathbb{E}_x \ell_D(D_\theta(x + O_\theta(x)), t) \tag{5.5}$$

where ℓ_D is the loss function used to train the D_θ, and $\mathcal{L}^D_{adv}$ loss encourages the perturbed samples to be misclassified as target class t (fake class label (0)).

Similar to the work on Carlini et. al. [60], we add a soft hinge loss on the L_2 norm to bound the magnitude of the perturbation as follows :

$$\mathcal{L}_{hings} = \mathbb{E}_x \max(||O(x)||_2 - \epsilon, 0) \tag{5.6}$$

where ϵ represents for a user-defined bound. Finally, the full objective of O_θ can be expressed as:

$$\mathcal{L}_{\mathcal{O}} = \mathcal{L}_{adv}^{D} + \alpha \mathcal{L}_{hings} \tag{5.7}$$

where α is used to regulate the relative importance of the $\mathcal{L}_{hings}$, and $\mathcal{L}_{adv}^{D}$ is used to produce adversarial samples, maximizing the attack success rate. Now, the defender has two networks G_θ and O_θ which try to minimize the detection success of the attacker and form a minimax game between the attacker and the defender.

5.3.2 Attacker

The attacker is a classifier would be a neural network that learns to classify input data- real, fake, and adversarial samples. The architecture of the attacker is similar to the Discriminator model of GAN, except the discriminator learns from three types of data. Therefore, in this model, we consider the discriminator as the attacker D_θ, where D_θ's aim to maximize the probability of assigning the samples in the correct class. In the case of adversarial samples x^{adv} generated by O_θ, the D_θ's learning is to provide a higher probability that the x^{adv} comes from the real distribution. Simultaneously, the defender (G_θ and O_θ) tries to minimize D_θ's detection probability by generating fake and adversarial samples. Therefore, the interaction between the defender and the adversary forms a minimax game and the value function $V(G_\theta, O_\theta; D_\theta)$ of the game can be defined as follows:

$$\begin{aligned}\min_{G_\theta, O_\theta} \max_{D_\theta} V(G_\theta, O_\theta; D_\theta) =& \mathbb{E}_{x \sim P_{real}(x)} \left[\phi(D_\theta(x))\right] + \mathbb{E}_{x \sim P_{real}(x)} \left[\phi(D_\theta(x + O_\theta(x)))\right] \\ &+ \mathbb{E}_{z \sim P_z(z)} \left[\phi(1 - D_\theta(G_\theta(z)))\right]\end{aligned} \tag{5.8}$$

where ϕ is the measuring function, and the best strategy for the model is when the discriminator provides output 1/2 for each $x, x^{'}$ and x^{adv}. In the above game model, the defender aims to minimize its maximum possible loss against the best policy of the adversary. Therefore, we can rewrite the equation (5.8) from the game-theoretic perspective as follows:

$$\pi^*, \rho^* = arg \min_{p \sim \pi} \max_{r \sim \rho} \mathbb{E}_{\pi,\rho}[J(M_p(\theta), \chi, \mathbb{Y})] \tag{5.9}$$

in Equation (5.9) the defender tries to maximize the loss of the attacker by generating fake and adversarial examples under a strategy ρ where the attacker's goal is to take a strategy π that minimizes his expected loss of detecting fake and real samples by changing model parameters θ to a new $M_p(\theta)$. π^*, ρ^* represents the best response for the attacker and the defender, respectively.

In practice, traditional GAN training sometimes leads to mode dropping, where the model can collapse due to the differences of the minimax and maximin solutions hypothesized. Also, optimizing D_θ to completion in the inner loop of training is computationally prohibitive, and on finite datasets, it would result in overfitting. But alternatively, optimization of D_θ in k steps and G_θ in one step provides the approximate optimal solution. Similarly, we train three networks simultaneously in our model, alternating between k steps of optimizing D_θ, one step of optimizing O_θ, and s steps of optimizing G_θ to stable the model from D_θ's rejecting samples with high confidence since they differ from the training data. Figure (5.2) shows some fake and adversarial samples generated by the TS-GAN model.

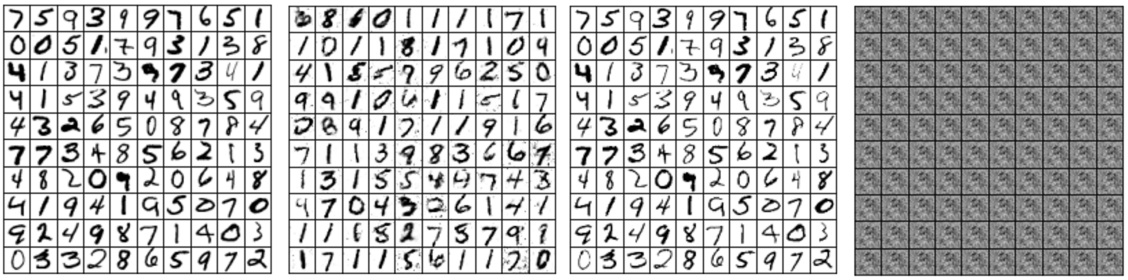


Figure 5.2: (a) MINST real digit samples for training (b) Fake digit samples generated by G_θ (c) Adversarial digit samples generated by using O_θ and real digit (d) Perturbations created by using O_θ

5.4 Experiments

This section investigates our method's empirical results and benefits on the MNIST dataset [61], where each sample in the dataset is 28×28-sized images of hand-written digits. We used 18000 samples from the dataset as real data, while the generator generated the fake data. We used (1024, 512, 256) hidden units for the Discriminator model, (256, 512, 1024) hidden units for the Generator model, and (256, 512, 1024) hidden units for the Obfuscator model to train three neural networks simultaneously where a rectified linear function is applied in each layer. We used k = 1 and s = 2 for the optimization steps. We also set the perturbation to have a magnitude between [-0.3, 0.3] in each of its components applied each step of training of Obfuscator to generate adversarial examples. Then, we train the networks using batches of size 32 for 300 epochs. We compare the performance of TS-GAN and the GAN model, where the standard GAN model is considered as a one-sided feature manipulation and TS-GAN as a two-sided feature manipulation binary classification problem. Finally, we took the MINST test dataset, which comprises 10000 samples for real class labels, and used a standard GAN with modified architecture and training data samples to produce another 10000 samples for fake class labels.

5.4.1 Measuring the Similarity of Features

Our first experiment investigates the impact of varying the similarity between FSG game model and TS-GAN model. We use Earth Mover Distance (EMD) [47, 53] as a similarity metric in the FSG game model, where this game considers two features in each object. In addition, for both real and fake objects, we utilize the importance value 1 and set the feature modification cost to (0.25, 0.1). In Figure 5.3 (a), the first observation in the FSG model is that two-sided deception has an advantage in highly informative cases, and the utility of the defender grows more as the feature distributions become more informative (higher EMD). In the TS-GAN model Figure 5.3 (b), we see a similar pattern: the attacker loss is higher in two-sided feature manipulation than in one-sided feature manipulation.

Fake image quality isn't good enough in the early stages of learning, resulting in a larger gap between real and fake samples. As a result, the attacker loss in the TS-GAN model is higher. Because TS-GAN consists of two generative models that we consider as the defender, we examine attacker loss rather than defender gain. However, after 150 epochs, the loss of the attacker decreases.

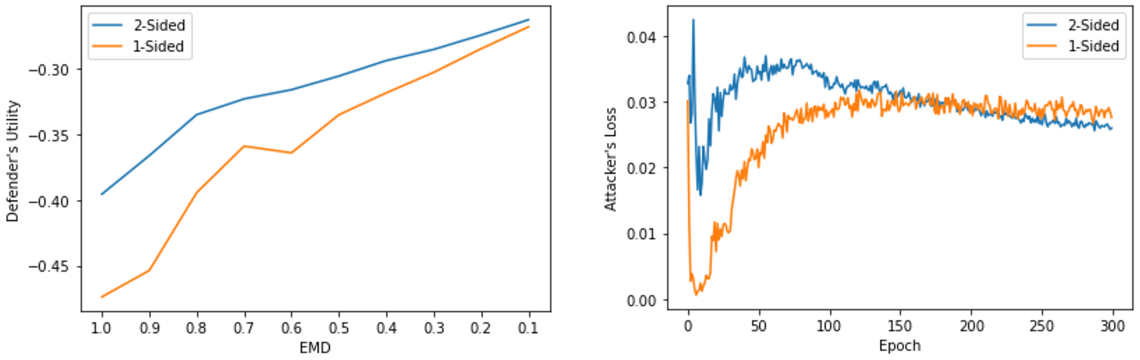


Figure 5.3: (a) Comparison of defender's utility in FSG game when modifying features of both real and fake objects (two-sided) vs. only modifying features of fake objects (one-sided) (b) Comparison of the attacker's loss between TS-GAN (two-sided) and GAN (one-sided)

Table (5.1) shows different evaluation measures [62] used to compare the overall performance of the discriminator between TS-GAN and GAN methods. When the discriminator gives output greater than or equal to 0.5 on the test data, we consider predicting a real class label in binary classification, otherwise a fake class label. The results show that the discriminator of TS-GAN outperforms traditional GAN considered in detecting real and fake samples. The results show that the discriminator of TS-GAN outperforms traditional GAN in detecting real and fake samples. The attacker (discriminator) learns in the TS-GAN model over time from real, fake, and adversarial data, making the attacker more robust in distinguishing real and fake samples.

Table 5.1: Comparison of the discriminator's performance between TS-GAN and GAN

	Accuracy	F1-Score	Precision	Recall
GAN	32.1 %	55.8 %	48.5 %	65.9 %
TS-GAN	59.4 %	75.9 %	73.5 %	78.5 %

5.4.1.1 Effectiveness of TS-GAN in Semi-Whitebox Settings

This experiment explores a possible way of using the adversarial examples of our TS-GAN model where the Obfuscator creates the perturbations based on binary classification. Figure 5.2 (d) shows some random perturbation generated by the Obfuscator. We applied the adversarial examples in a semi-whitebox settings to study the effectiveness of the adversarial attack where we assume that the adversary knows the training data source only. The target model is a three-layer neural network with 1024, 512, and 256 hidden units, with each layer using a rectified linear function with 0.3 dropouts. We trained the target model with MINST dataset disjoint from the TS-GAN used dataset for 100 epochs where the dataset contains 10 categorical classes, 0-9 digits.

We compare our attack by evaluating the success rate of fooling the target neural network. As a baseline comparison, we used the fast gradient sign method (FGSM) and projected gradient descent (PGD) attacks, which assume that the adversary is completely aware of the target model's architecture and training data source. For FSGM, we set $\epsilon = 0.5$ and for PDG, we select $\epsilon = 0.1$ and search for $1e4$ steps with 1000 iterations. Despite the fact that our model is trained on binary classification, the findings in Table (5.2) show that our model gives better attacks for classes 0, 1, and 3 than the PGD model. The attack ratio, on the other hand, is not as good as the FSGM model. But, our model successfully generates adversarial examples for all classes while comparing to the test accuracy of the target model.

Table 5.2: Adversarial attack success rate per class, the lower is better (the row labeled 'NN test' refers to the target model's test data accuracy, not any attack measure)

Class	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
NN test	97.8	98.4	96.0	96.3	96.7	96.8	96.9	95.7	93.7	96.5
FGSM	1.0	0.0	0.0	0.0	0.0	34.5	15.0	11.3	4.3	11.2
PGD L_∞	92.7	90.9	63.4	68.9	75.7	71.9	77.6	73.4	55.0	61.4
TS-GAN	**87.1**	**77.0**	97.2	**60.8**	91.7	87.4	88.1	87.1	76.9	94.1

5.5 Related Works

Several prior articles have discussed the concept of two-sided deception where adversaries are confused by both real-looking fake objects and fake-looking real objects. Rowe et al. show that adopting two-sided deception improves defense by terrifying attackers away [43]. De Gaspari et al. introduced a proof-of-concept prototype system in which the production system is equipped with active defense capabilities [45].The honeypot framework is another powerful deception strategy for bolstering defenses because it integrates actual hosts, honeypots, and fake-looking honeypots [44]. Caroll and Grosu introduced the signaling deception game to determine the optimal equilibria strategies of sending signals that improve the honeypot's deception [6]. These studies, however, do not take into account feature deception, in which individual features can be manipulated and revealed to the attacker.

Feature deception has been studied in many domains, primarily focusing on using cyber security to conceal essential pieces of information. Network adversaries use tools like Nmap to probe and analyze various target properties, but deliberately manipulating probe reply packets can confuse and foil an attack effort [63, 64, 65]. Masking important features of a real object make more difficult to detect, whereas adapting fake objects can lure adversaries by strategically revealing fake information [66, 67]. Shi et al. introduce the

feature deception game to optimally abstract decision-relevant information as features of network systems where an adversary observes the features of each network system and then could choose a particular system to attack [44]. Aggarwal et al. use HackIT to change the observable features of fake and real systems, creating confusion for attackers and improving cyberdefense when the attackers are human [52]. Miah et al. introduce the Feature Selection Game to determine the optimal strategy for the defender to alter observable characteristics of both real host and decoys so that the attackers can not distinguish them apart [53]. We study two-sided feature deception using adversarial machine learning to scale large-size feature manipulation problems.

The most well-known adversarial learning approach is Generative Adversarial Networks (GAN) [51] where two neural networks are used to generate deceptive inputs and detect differences between real and fake inputs. Some studies have suggested using multiple generators and simultaneously training them help to focus on some modes of the data space and learn better [68, 69]. However, the generator models in various GAN primarily focus on generating fake samples that resemble real inputs while not manipulating real samples. Many strategies are now prevalent to manipulate real inputs and generate adversarial examples, in which carefully chosen perturbations to input data can result in high-probability misclassification [50, 60, 70, 71, 72]. Goodfellow et al. introduce the fast gradient sign approach, which applies a first-order approximation of the loss function with respect to the input data and injecting perturbed data into their training dataset [55]. Xiao et al. use generative adversarial networks to generate adversarial examples that can learn and approximate the distribution of original instances [73]. In our work, we simultaneously use adversarial networks to generate fake and adversarial samples that can be used to solve two-sided deception on a larger scale.

5.6 Discussion and Further Application

The TS-GAN model introduces a new way of creating various decoys and camouflages. Though the experiments of this work are based on image datasets where the features are gray-scale pixel values, the model is not confined to that type of data only. This model can generate fake samples of any type of data with properties similar to real training samples. Simultaneously, it can identify the strategy of modifying the characteristics of real samples to make detection more difficult.

One practical application of this model could be to prevent statistical traffic analysis where an adversary can classify different applications and protocols from the observable statistical properties, especially from the meta-data (e.g., packet size, timing, flow directions, etc.). Network traffic obfuscation is a technique where network traffic is manipulated (e.g., adding dummy bytes with the packets to increase packet size) to limit the attacker's gathering of information by causing errors in the classification models. This obfuscation approach is effective at reducing the risk of passive reconnaissance, where an attacker gathers traffic and uses statistical analysis to categorize different patterns (e.g., protocols, applications, user information, etc.). Also, the network administrators can send dummy packets that reveal different types of fake applications and protocol information to introduce uncertainty to the network. Our model can generalize the overall deception approach by creating dummy packets and obfuscating real traffic packets. Another application could be generating honeyflows which is applicable to reveal the various type of fake vulnerability (e.g., os vulnerability, service vulnerability, etc.). An adversary can analyze network packets' features, such as TTL, Window size, Data Fragment, or banner information, to figure out specific operating system versions or service information. Obfuscating these features and generating honey packets can delay the detection of real vulnerabilities. Therefore, we can use these datasets to test our model and analyze more practical experiments.

5.7 Conclusion

Two-sided feature deception is an effective strategy for the defender for generating cyber camouflages. Honeypot Feature Selection Game (HFSG) can provide the optimal solution for creating real and fake samples, but it's only suitable for handling when the sample size is small. In this paper, we propose TS-GAN, a novel algorithm that can generate real-like fake samples and fake-like real samples, potentially increasing the difficulty of detecting real and fake samples for an adversary. Our algorithm can solve large and complex two-sdied feature deception problem and provide approximation solution of the minimax two-sided deception game. Moreover, the TS-GAN can be used as a robust binary classifier and generating successful adversarial examples.

There are a number of ways that this work could be extended, including the fact that the HFSG model is a one-shot game, therefore the attacker and defender's learning cannot be fully represented. The HFSG game can be modeled in a multi-round environment and compared to TS-GAN learning. We could also use a different dataset or architecture for measuring the performance of TS-GAN algorithm. However, our main focus in this work is to develop a scalable algorithm that can solve large and complex two-sided feature deception problems, not to provide better performance measures. Therefore, this solution is sufficient to solve the problem that successfully meets our goals.

Chapter 6

Vulnerability-Driven Decoy Traffic Optimization

6.1 Introduction

Advanced Persistent Threats (APTs) are a significant concern for enterprises. In such scenarios, advanced adversaries take slow and deliberate steps over months and even years to compromise critical resources (e.g., workstations and servers) in a network. A key step in the kill chain of APTs is reconnaissance. Historically, reconnaissance is largely active, for example using network port scanning to identify which hosts are running which services. In response, many enterprises closely monitor their networks for scanning attacks.

Simultaneously, Software Defined Networking (SDN) technology is emerging as a powerful primitive for enterprise network security. SDN offers a global perspective on network communications between hosts. It can be used as an enhanced tool to identify network scanning, provide flexible access control to mitigate attackers bypassing defenses such as firewalls, and even prevent spoofing. However, the increased functionality within network elements (e.g., switches) makes them a target for attack. A compromised SDN switch is particularly dangerous, because it can perform reconnaissance passively. As a result, defenders may have little to no signal that an APT is in process.

In our previous work, we discussed how the defenders can generate fake (honey) traffic to support honeypots or to conceal the properties of real traffic on their networks by modifying various features(e.g packet size, timing, etc.) of a network packet for both real and fake traffic. This work addresses the threat of passive network reconnaissance and pro-

poses an application to use honey traffic: fake flows deceptively crafted to make a passive attacker think specific resources (e.g., workstations and servers) exist and have specific un-patched, vulnerable software. We consider that the adversary knows about the possibility of honey traffic and uses game theory to characterize how best to send honey traffic. For doing so, we demonstrate how a defender can either successfully deflect or deplete an adversary using optimal amount of honey traffic.

We model this defender-attacker interaction as a two-player non-zero-sum Stackelberg game. In this game, the defender sends honey traffic to confound the adversary's knowledge. However, if the defender sends too much honey traffic, the network may become overloaded. In contrast, the adversary wishes to act on information obtained using passive reconnaissance (e.g., a banner string indicating a server is running a vulnerable version of Apache). However, if the adversary acts on information in the honey traffic, it will unknowingly attack an intrusion detection node and be discovered. Thus, the game presents an opportunity to design an optimal strategy for defense. Our algorithmic solution finds the optimal strategy for deploying honey flows that is fast enough to be used for realistic networks. Also, we present an empirical evaluation of the performance of our game model solutions under different conditions, as well as the scalability of the algorithm.

6.2 Motivation and Related Work

Enterprise network administrators are increasingly concerned with Advanced Persistent Threats (APTs) where adversaries first obtain a small foothold within the network and then stealthily expand their penetration over the course of months, sometimes years. The past decade has provided numerous examples of such targeted attacks, e.g., Carbanak [74], OperationAurora [75]. Such attacks require significant planning. Initially, adversaries identify attack vectors including 1) vulnerable servers or hosts, 2) poorly configured security protocols, 3) unprotected credentials, and 4) vulnerable network configurations. To do so, they leverage network protocol banner grabbing, active port scanning, and passive

Table 6.1: Example of types of information used for attacks

Target Type	**Analysis Space**	**Examples**
Fingerprinting OS	TTL, Packet Size, DF Flag, SackOk, NOP Flag, Time Stamp	Windows 2003 and XP
Server software, version, service type	Default banners	Apache HTTP 2.2, Windows Server 2003
Network topology, forwarding logic	Flow-rule update frequency, controller-switch communication	Lack of TLS adoption, modified flow rules
Employee Credentials, personal information	Server-client traffic header and data	HTTP traffic, HTTPS traffic with weak TLS/SSL

monitoring [76, 77]. Examples of different types of desired information and corresponding attacks are shown in Table 6.1.

SDN has the potential to address operational and security challenges large enterprise networks [78]. They provide flexibility to programmatically and dynamically re-configuring traffic forwarding within a network [79] and provide opportunities for granular policy enforcement [80]. However, these more functional network switches form a large target for attackers as they can provide a foothold to perform data plane attacks using advanced reconnaissance and data manipulation and redirection [81]. Using one or more compromised switches, an adversary can learn critical information to mount attacks, including network topology and software and hardware vulnerabilities [82, 83].

Deception is an important tactic against adversary reconnaissance, and there have been a variety of different approaches that apply game-theoretic analysis to cyber decep-

tion [84]. Many of these previous works have focused on how to effectively use honeypots (fake systems) as part of a network defense [6, 85, 33, 9]. This has included work on signaling games where the goal is to make real and fake systems hard to distinguish [53]. Work on security games (including games modeling both physical and cybersecurity) focuses on deception to manipulate the beliefs of an attacker [86, 87, 65, 88]. Another research [12] proposes a game model of deception in the reconnaissance phase of an attack, though they do not consider honey flows. Stackelberg game models have been used to find optimal strategies for cyber-physical systems [89].

6.3 System Model Overview

Our proposed game-theoretic model is based on a deception system [90] that could mislead or delay the passive reconnaissance by an adversary. The system provides this deception using *honey traffic* that is precisely controlled by the defender. Traditionally, a network flow is defined as a 5-tuple: source IP, source port, destination IP, destination port, and protocol (e.g., TCP). For simplicity, we assume honey flows include network flows in both directions to simulate real network communication.

Honey flows can fake information in network flow identifiers. For example, a honey flow can attempt to make the adversary believe a non-existent host has a specific IP address, or a host is running a server on a specific port. Due to the flexible packet forwarding capabilities of SDN, the defender can route honey flows through any path it chooses, e.g., to tempt an adversary that has compromised a switch on a non-standard path. Honey flows can fake information in the packet payload itself. For example, network servers often respond with a banner string indicating the version of the software, and sometimes even the OS version of the host. Attackers often use this banner information to identify unpatched vulnerabilities on the network. Honey flows can simulate servers with known vulnerabilities, making it appear as if there are easy targets. If at any point the adversary acts on this information (i.e., connects to a fake IP address), the system will redirect the traffic to

an intrusion detection node. Since the intrusion detection node does not normally receive network connections, the existence of any traffic directed towards it indicates the presence of an adversary on the network.

6.3.1 Threat Model

The adversary's goal is to compromise networked resources, e.g., workstations and servers, without detection. The adversary does not know what hosts are on the network or which hosts have vulnerabilities. It must discover vulnerable hosts using network reconnaissance. The adversary assumes the defender has deployed state-of-the-art intrusion detection systems that can identify active network reconnaissance such as network port scanning. However, we assume the adversary has gained a foothold on one or more network switches (an upper-bound of which is defined by the model). Using this vantage point, the adversary is able to inspect all packets that flow through the compromised switches. In doing so, it can learn (1) network topology and which ports servers are listening to by observing network flow identifiers, (2) about the installed software versions by observing server and client banner strings. We assume the adversary can map between banner strings and known vulnerabilities and their corresponding exploits. We conservatively assume this mapping can occur on the switch, or can be done without the knowledge of the defender. The adversary also has the capability to initiate new network flows from the switch while forging the source IP address, as response traffic will flow back through the compromised switch and terminate as if it was delivered to the real host. Finally, we assume the adversary is rational and is aware of the existence of the deception system and that honey traffic may be sent to fake hosts. However, the adversary does not know the distribution of honey traffic.

We assume the defender's network contains real hosts with exploitable vulnerabilities. The defender is aware of those. For example, the defender's inventory system may indicate the existence of an unpatched and vulnerable server, but due to production requirements, the server is not yet patched. We further assume the defender can identify valuations of

each network asset and approximate the valuation of the assets to the attacker (e.g., domain controllers that authenticate users are valuable targets).

6.4 Game Model

An important question we must answer to deploy deceptive system effectively is how to optimize the honey traffic created by the system, including how much traffic to create of different types. This decision must balance many factors, including the severity of different types of vulnerabilities, their prevalence on the network, and the costs of generating different types of honey flows (e.g., the added network congestion). In addition, a sophisticated APT attacker may be aware of the possible use of this deception technique, so the decisions should robust against optimal responses to honey traffic by such attackers. Finally, we note that many aspects of the environment can change frequently; for example, new zero day vulnerabilities may be discovered that require an immediate response, or the characteristics of the real network traffic may change. Therefore, we require a method for making fast autonomous decisions that can be adjusted quickly.

We propose a game theoretic model to optimize the honey flow strategy for deception. Our model captures several of the important factors that determine how flows should be deployed against a sophisticated adversary, but it remains simple enough that we can solve it for realistic problems in seconds (see Section 6.5 for details) allowing us to rapidly adapt to changing conditions. Specifically, we model the interaction as a two-player non-zero-sum Stackelberg game between the defender (leader) and an attacker (follower) where the defender plays a mixed strategy and the attacker plays pure strategy. This builds on a large body of previous work that uses Stackelberg models for security [91], including cyber deception using honeypots [33].

Table 6.2: Game Notation

d	defender
a	attacker
$V_i \in V$	set of i types of vulnerabilities in the network
R_i	number of real flows indicating V_i
H_i	upper bound on the number of honey flows indicating V_i
Φ_{ij}	action of selecting $j \in [0, H_i]$ honey flows for V_i
Φ	defender's mixed strategy as the marginal probabilities over $\{\Phi_{i0}, \ldots, \Phi_{iH_i}\}$
C_i	cost of creating each honey flow that indicates V_i
$\upsilon_i^{a,r}$ $\upsilon_i^{a,h}$	the value the attacker gains from attacking a real or fake flow of type V_i
$\upsilon_i^{d,r}$ $\upsilon_i^{d,h}$	the value the defender loses from an attack against a real or fake flow of type V_i
a_i	denotes the action of attacking a flow of type V_i where a_0 is the no-attack action, yielding 0 payoff

We now formally define the strategies and utilities of the players using the notation

listed in Table 7.1. We assume that the defender is using a deceptive system as a mitigation for a specific set of i vulnerabilities that we label V_i. Every flow on the network indicates the presence of at most one of these types of vulnerabilities in a specific host. The real network traffic is characterized by the number of real flows R_i that indicate vulnerability type V_i. The pure strategies for defender are vectors that represent the number of honey flows that are created that indicate each type of vulnerability V_i; we write Φ_{ij} to represent the marginal pure action of creating j flows of type V_i. These fake flows do not need to interact with real hosts; they can advertise the existence of fake network assets (i.e., honeypots). The defender can play a mixed strategy that randomizes the number of flows of each type that are created, which we denote by Φ. To keep the game finite we define the maximum number of flows that can be created of each type as R_i. The attacker's pure strategy a_i represents choosing to attack a flow of type V_i, or not to attack. We assume the attacker cannot reliably distinguish real flow and honey flow, so an attack on a specific type corresponds to drawing a random flow from the set of all real and fake flows of this type.

The utilities for the players depend on which vulnerability type the attacker chooses, as well as on how many real and honey flows of that type are on the network. An attack on a real flow will result in a higher value for the attacker than on a honey flow of the same type, and vice versa for the defender. Specifically, if the attacker chooses type V_i, it gains a utility $\upsilon_i^{a,r}$, which is greater than or equal to the value for attacking a honey flow of the same type $\upsilon_i^{a,h}$ (which may be negative or 0). We assume that this component of the utility function is zero-sum, so the defender's values are $\upsilon_i^{d,r} = -\upsilon_i^{a,r}$ and $\upsilon_i^{d,h} = -\upsilon_i^{a,h}$. The defender's utility function includes a cost term C_i that models the marginal cost of adding each additional flow of type V_i (for example, the additional network congestion which can vary depending on the type of flow). If the defender plays strategy Φ and the attacker attacks the V_i, the defender's expected utility is defined as follows:

$$U^d(\Phi, i) = P_i^r \upsilon_i^{d,r} + (1 - P_i^r)\upsilon_i^{d,h} - C^h \tag{6.1}$$

Here, P_i^r denotes the probability of attacking a vulnerability of type V_i which can be calculated as follows:

$$RSMN(mc, P) = \sum_{j \in \{0,\ldots,H_i\}} \Phi_{ij}(R_i/(j + R_i))$$

The overall cost C^h for playing Φ is given by Equation 7.1:

$$C^h = \sum_{i \in V} \sum_{j \in \{0,\ldots,H_i\}} (\Phi_{ij} \times j \times C_i)$$

Analogously, for the attacker the expected utility is given by:

$$U^a(\Phi, i) = P_i^r \upsilon_i^{a,r} + (1 - P_i^r)\upsilon_i^{a,h} \tag{6.2}$$

6.4.1 Game Example

Consider a network with two types of vulnerabilities. Let the values be $\upsilon^{a,r} = (10, 20)$, and $\upsilon^{a,h} = (-5, -10)$, and the cost of creating honey flow indicating each type of vulnerability is $\mathrm{C} = (1, 0.5)$. The total number real flows indicating each type of real vulnerability is $\mathrm{R} = (5, 5)$, and the upper bound on honey flows is $\mathrm{H} = (2, 3)$. Thus at most two honey flows of 1^{st} vulnerability type and three honey flows of 2^{nd} vulnerability type can be created. Now, consider if the defender plays the following strategy Φ:

$\Phi =$ { $\Phi_1 =$

Φ_{10}	0
Φ_{11}	0.5
Φ_{12}	0.5
-	-

, $\Phi_2 =$

Φ_{20}	0
Φ_{21}	0
Φ_{22}	0
Φ_{23}	1

}

In Φ the defender creates one honey flow 50% of the time and two honey flows 50% of the time with type 1 vulnerability. The defender also creates three honey flows 100% of the time type 2 vulnerability. The attacker's best response is to attack vulnerability type 2 with expected utility $\mathrm{U^a}(\Phi, 2) = 8.75$, and the defender utility is $\mathrm{U^d}(\Phi, 2) = -11.75$.

6.4.2 Optimal Defender's Linear Program

Our objective is to compute a Stackelberg equilibrium that maximizes the defender's expected utility, assuming that the attacker will also play the best response. To determine the equilibrium of the game, we formulate a linear program (LP) where the attacker's pure strategy a is a binary variable. We create a variable for each defender's pure strategy $\Phi_{i,j}$, the action of creating j honey flows for V_i. The following LP computes the defender's optimal mixed strategy for each type of vulnerability under the constraint that the attacker plays a pure-strategy best response:

$$\max_{i \in V} \; U^d(\Phi, \mathrm{i})\ \mathrm{a_i} \tag{6.3}$$

$$s.t. \quad a_i \in \{0,1\}, \quad \Phi_{ij} \in [0,1]$$

$$U^a(\Phi, \mathrm{i})\ \mathrm{a_i} \geq \mathrm{U^a}(\Phi, \mathrm{i'})\ \mathrm{a_i} \quad \forall\ \mathrm{i}, \mathrm{i'} \in \mathrm{V} \tag{6.4}$$

$$\sum_{j \in \{0,\ldots,H_i\}} \Phi_{ij} = 1 \quad \forall \Phi_i \in \Phi \tag{6.5}$$

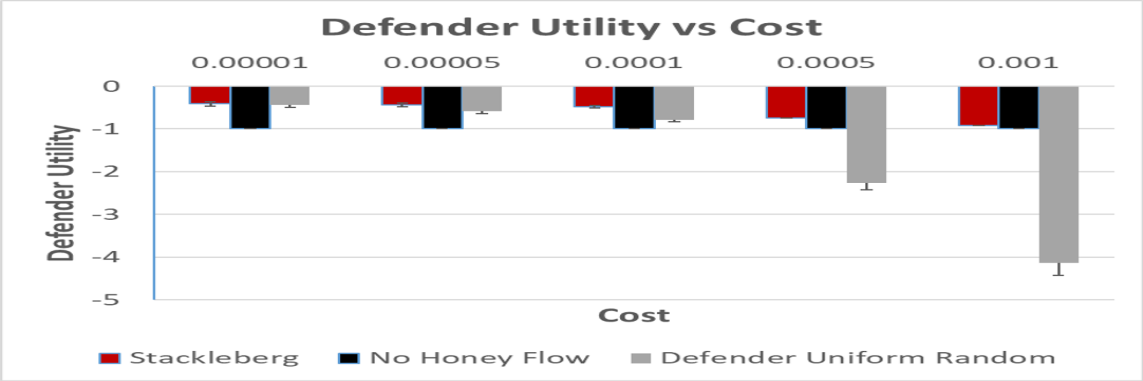
$$\sum_{i \in V} a_i = 1 \tag{6.6}$$

In the above formulation the unknown variables are the defender's strategy $\{\Phi_{\mathrm{i0}}, \ldots, \Phi_{\mathrm{iH_i}}\}$ for each $\Phi_i \in \Phi$ and the attacker's action a_i. Equation 7.3 is the objective function of the LP that maximizes the defender's expected utility. The inequality in Equation 7.4 ensures that the attacker plays a best response. Finally, Equation 7.6 forces the defender strategy to be a valid probability distribution.

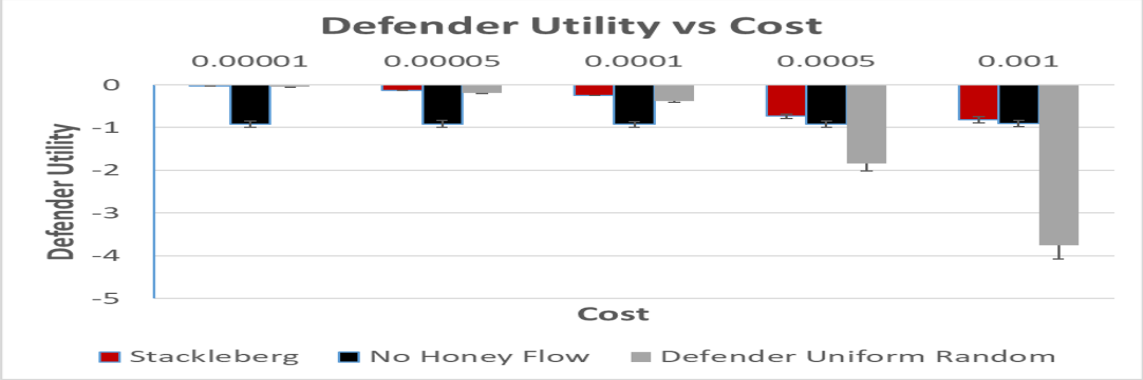
6.5 Experiments and Solution Evaluation

Our experiments focuses on evaluating the solution quality of the proposed Stackelberg game model for optimizing honey flows compared to some plausible baselines, 1) not generating honey flows at all, 2) using a uniform random policy for generating honey

flows. We average the results over 100 randomly generated games, each with 5 types of vulnerabilities. We set the number of real flows for each type to 500 and the upper bound on the number of honey flows for each type is uniform randomly generated from [500,1000]. Values are described in the caption, and we vary the costs of creating flows as shown in the Figure 7.4.



(a)



(b)

Figure 6.1: Comparison of defender utility when the defender uses different values: a) the value of attacking a fake vulnerability is zero and a real is 1 b) the value of attacking a fake vulnerability is the same as real value and the values are randomly generated from [0.5, 1.0].

The results in Figure 7.4 show that the game theoretic solution significantly outper-

forms the two baselines in most settings, demonstrating the value of optimizing the honey flow generation based on the specific scenario. We also note that the cost has a significant impact on the overall result; with a high cost the game theories solution is similar to not generating flows at all (since they are not very cost effective). Random honey flow generation can be detrimental for the defender. With a low cost, the performance of the game theoretic solution is similar to the uniform random policy; since flows are so cheap it is effective to create a very large number of them without much regard to strategy. With intermediate costs the value of the strategic optimization is highest, which is the most likely scenario in real applications.

In our second experiment we consider vulnerabilities with different values and examine the variation in the optimal solution as we vary the number of real flows. We use 5 vulnerabilities with the values of the real systems $(0.8, 0.5, 0.9, 0.6, 1.0)$ and attacking any fake system gives 0. The cost is 0.0005 for all types. The results in Figure 6.2 show that the defender's strategy is to create more honey flows for the high valued vulnerabilities. As the number of real flows increases the cost of adding flows to create a high ratio is substantial and the overall number created drops for all types.

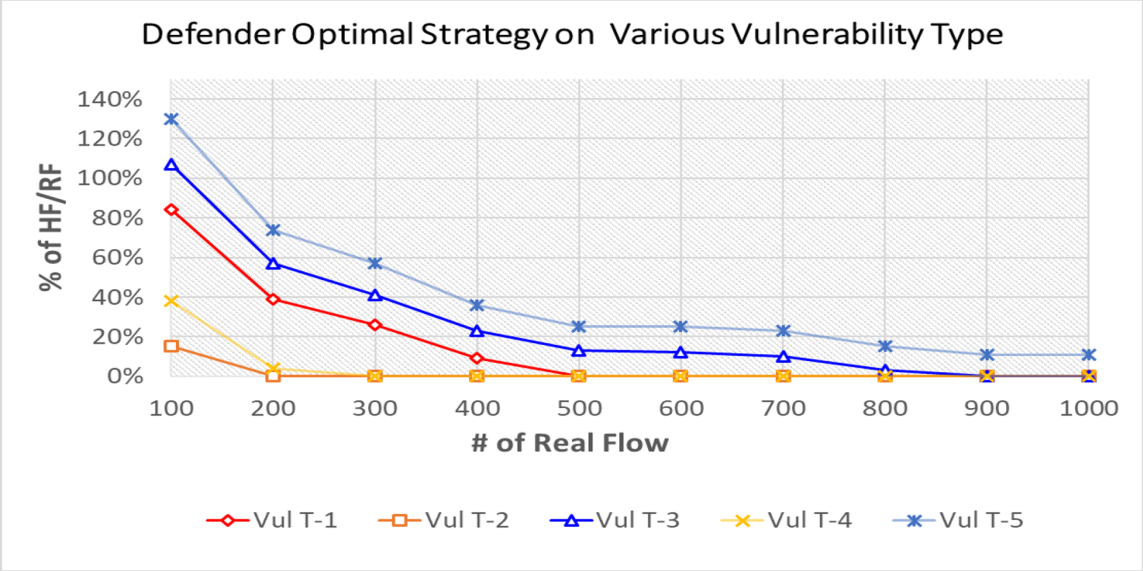


Figure 6.2: Defender's optimal strategy as number of real flows varies.

6.5.1 Solution Analysis

We now analyze how the ratio of honey flows to real flows changes in the optimal solution as we change the number of real flows. In Figure 6.3, the network setup consists of four vulnerabilities with values of (10, 20, 30, 40) and fake flows with values of (9, 18, 27, 32). The cost of generating each honey flow is 0.1. We show the defender's expected utility as we increase the ratio of honey flows to real flows. Each line represents a different number of real flows in the original game. We see that the defender utility increases as we add honey flows, but only up to a point; when the marginal value is less than the cost the optimal solution is to stop adding additional flows. We see this in the shape of the curves.

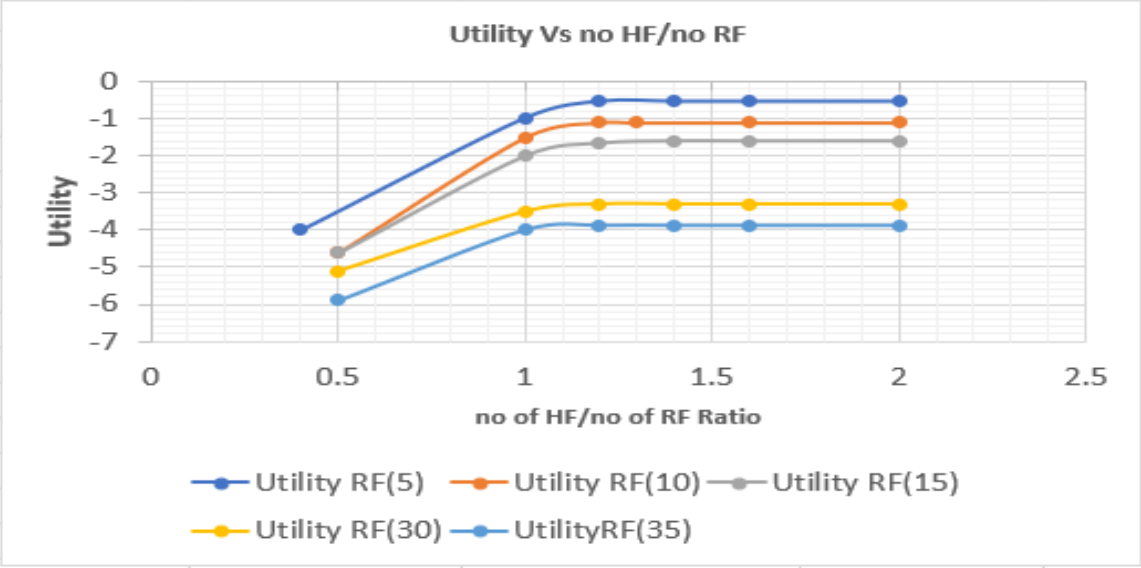
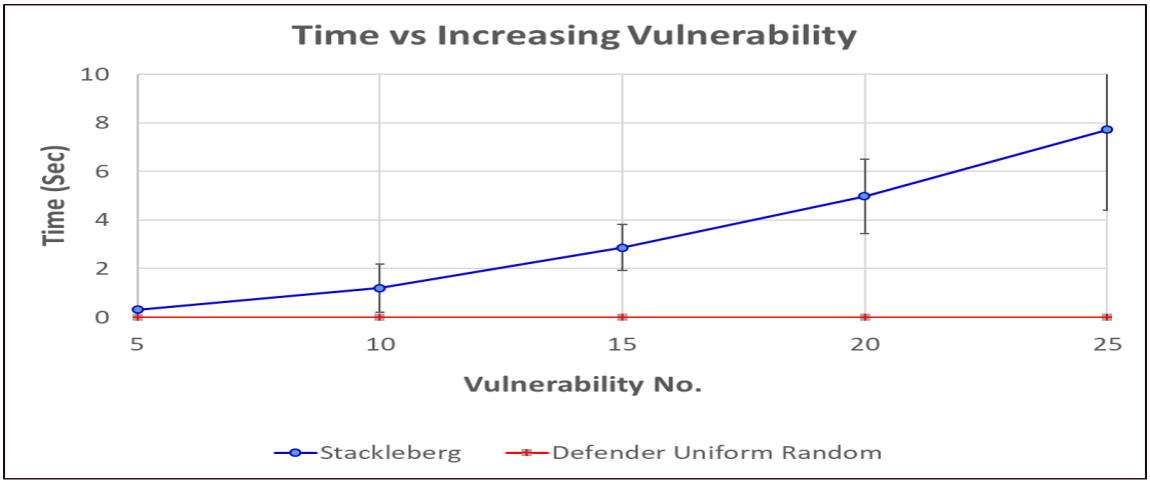


Figure 6.3: Utility with varying honey flow ratios

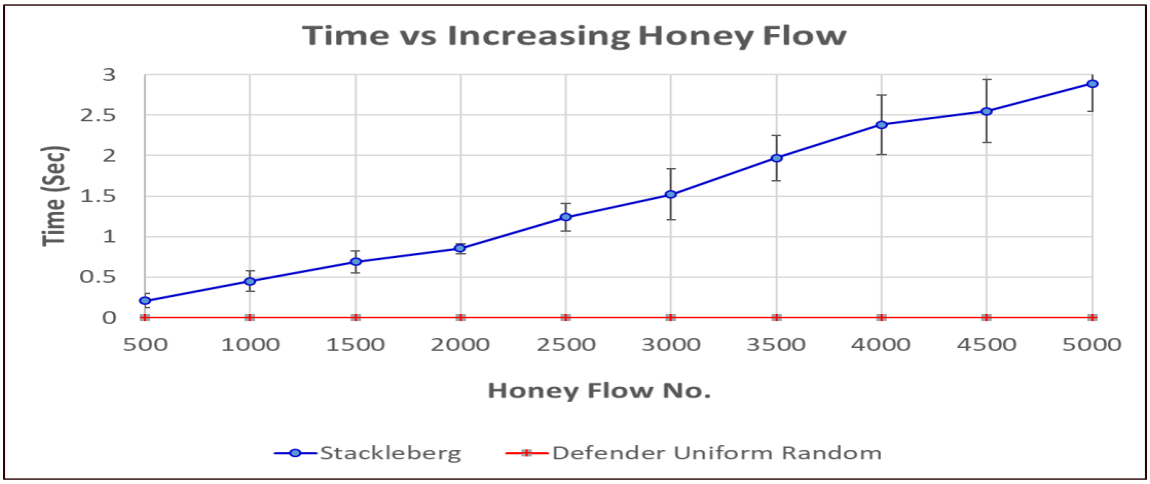
6.5.2 Scalability Evaluation

In a practical application of this model we would need to be able to calculate the optimal strategy quickly, since the network may change frequently leading to different game parameters. For example, the number of real flows will change over time, as will the hosts in the network. The values of traffic and vulnerabilities, and the specific vulnerabilities we

are most interested in can change also (e.g., due to the discovery of new vulnerabilities). We evaluate the scalability of the basic LP solution for this game as we increase the size of the game in two key dimensions: 1) increasing the number of vulnerability types, and 2) increasing the number of flows.



(a)



(b)

Figure 6.4: Comparison of computational time when a) varying the number of vulnerabilities; b) varying the number of honey flows

We randomly generate games holding the other parameters constant to evaluate the

solution time. The results are shown in Figure 6.4. Though the solution time increases significantly as we increase the complexity of the game, we were able to solve realistic size games with a large number of flows and vulnerability types of interest within just a couple of seconds using this solution algorithm. This signifies that we can apply this to optimize honey flows in realistic size networks with a fast response rate; with further optimization we expect that the scalability could be improved significantly beyond this basic algorithm.

6.6 Conclusion

In this work, we use deceptively crafted honey traffic to confound the knowledge gained by adversary through passive network reconnaissance. We defined a Stackelberg game model for optimizing the quantity and type of honey flows to create. This model balances cost and value trade-offs in the presence of a sophisticated attacker, but can still be solved fast enough to be used in a dynamic network environment. We have evaluated this model by simulations that explore the properties of the game theory solutions.

There are a number of ways that this model could be improved, including incorporating network structure and variable host values into the analysis, and allowing for more overlap between vulnerabilities and flows (e.g., flows with more than one vulnerability). We can consider additional types of honey traffic, such as modifying real flows in deceptive ways. However, these may come with significantly increased computational costs for finding solutions, so we will need to develop faster algorithms to make solving these more complex models practical for a real implementation.

Chapter 7

Honeyflow Optimization based on Adversarial beliefs

7.1 Introduction

Cyber deception techniques using decoy objects can effectively reduce the quality of the hostile reconnaissance by providing false information to attackers. Specifically, honeypots are commonly applied to enhance network security. Network defenders install fake hosts (honeypots) to mislead attackers and make them hesitant to continue their additional attacks. They generate fake traffic to support honeypots or conceal natural traffic properties on their networks. Specifically, sending fake traffic can invalidate attackers' gathering information in the passive reconnaissance phase and confuses adversaries in identifying network vulnerabilities. However, generating fake traffic that mimics real traffic and operating them is challenging.

Recently, the Generative Adversarial Networks (GAN) [51] model has shown great promise in generating deceptive objects, focusing primarily on images and video applications. The model relies on pair of neural networks: generator and discriminator. Typically, the generator is trained to map from a latent space to a particular data distribution, and the discriminator distinguishes candidates produced by the generator from the true data distribution. The objective of the trained generator is to increase the error rate of the discriminator where the networks plays a zero-sum game in between them.

The previous chapter discussed how to deploy honeyflow to delay cyber attacks and a game model that provides the optimal solution for honey flow deployment. However,

we didn't go over how to make honeyflow or how the quality of honeyflow affects the attacker's decision-making. In this chapter, we considers the GAN-based model used to mimic a real network flow called honeyflow to mislead attackers. Specifically, honeyflows carry fake information, such as devices' vulnerabilities, in a network. To optimize the defender's honey traffic generation strategy while considering an adversary maintains varying beliefs about different types of real and honey flows, we present a two-player non-zero-sum Stackelberg game model. The attacker's belief reflects the defender's ability to create realistic deceptive honeyflow.

In the following sections, we will first discuss the network configuration briefly. Then we will discuss the basic deception architecture. After that, we will present the belief-based honeyflow game model. Finally, we will discuss the empirical results of the game-theoretic solution.

7.2 Network Configurations

We consider an SDN-based enterprise network consisting of servers, workstations, Openflow switches, routers, and logically or physically connected. In addition, the network includes fake hosts (e.g., honeypots) to enhance cyber security. The main objective of the honeypot is to attract attackers' attention and protect actual hosts. In the network topology, any host can be specified as a node in the network, where each node might have pre-existing vulnerabilities in OS and software. Nodes that have vulnerabilities are generally exploitable. The defender's inventory system may indicate the existence of an unpatched and vulnerable server, but due to production requirements, some devices are not yet patched. In addition, zero-day attacks are challenging to identify and prevent for the defender. Our model considers the *vulnerability value* to represent how difficult is to compromise a host for an attacker when the host contains a particular type of vulnerability. Every host in the network obtains a vulnerability value based on its actual situation, such as the OS type and application versions. For example, suppose a host deploys an

old version of OS or applications. In that case, it will get a relatively high vulnerability value, which means the attacker can exploit it with less effort or resources. Additionally, the same type of vulnerability in differently valued hosts can cause different costs to the defender. We assign an *importance value* to each host from the perspective of the defender and are determined by the type of devices (e.g., servers, laptops, and IoT devices) and the roles of the device owners (e.g., CEO or officer). For example, the server or database in the SDN network should be more valuable than a laptop that has no confidential information.

Assumption: We consider that the attacker attempts to identify targets based on the communications between clients and servers. Assume the attacker has control of at least one compromised switch and can only capture packets and execute necessary analysis during the reconnaissance phase. We assume the attacker has the necessary processing power to analyze its target information, including vulnerabilities and importance values, based on the observed network flows.

The trusted computing base (TCB) includes the system defender (SDN controller or a separate trusted server) and the southbound network between the SDN controller and SDN switches. We do not assume SDN switches are trustworthy, but we assume that not all the switches are compromised. We focus on passive traffic monitoring without discussing Openflow switch compromise methods and other malicious actions in the SDN environment. In addition, we only consider TLS-protected TCP communications. However, research shows that encrypted web traffic can leak information through packet length, packet timing, web flow size, and response delay.

7.3 Deception Architecture

A network administrator must monitor network data to keep the network running smoothly and efficiently. Therefore, monitoring and capturing specific traffics passing through the network, especially vulnerable servers, is vital before implementing honey flow. Currently,

sniffing tools are available to sort out vulnerable traffics and a defender's observation might include following:

- Normal Traffic Flows: This observation contains all the information that the SDN controller and network administrator can know, such as the flow rules and the amount of network flows.

- Honey Traffic: The defender has knowledge about all the information about the honey traffic, including the path, amount and contained misinformation of all the honey traffic.

- Partial Misbehavior: we assume if attacker trust the misinformation in honey traffic and deploy attacks based on the misinformation, the defender can detect them.

- System Security Situation: the defender can know the security situation that is introduced in the node model.

Based on the observations, the defender needs to determine the optimal deception strategies, including how much honey flow to develop and how many fake hosts to install. Generally, the network utilization is not near-maximum capacity during regular operation. However, exceeding honey traffic may cause congestion and cause network degradation.

To protect the valuable hosts in the network, the defender can deploy fake hosts (e.g., honeypot) to attract the attacker's attention. Meanwhile, one challenge is how to create natural enough honeyflows to mislead an attacker's behavior.

In an SDN environment, the SDN controller can obtain network flow information, such as source/destination hosts, the number of transferred packets and the bytes of packets. Leveraging on this feature, we consider the SDN controller to learn from the hosts and define network-level false information and application-level false information.

- Network-level information is applied to guide the honeypot to mimic regular hosts' behaviors, such as the frequency, length, and the number of network packets. Specifically, to mimic a regular host, we use six features to parameterize the creation of

honeyflow connections, which are (1) data first seen, (2) duration, (4) source/destination IP addresses and port numbers, (5) number of packets, and (6) bytes of each packet. To generate realistic honey connection, a honey connection generator needs to learn the pattern of a real server.

- Application-level information includes the details that assist attackers in searching for targets and prepare their exploitation. Specifically, attackers can generate its belief about one device's importance and vulnerabilities via these information. For example, the HTTP/HTTPS connections may include the server's OS information. Other application-level information involve Gmail cookies and passwords for unencrypted protocols.

GAN Based Honeyflow Generation : To mislead passive monitoring attacks, the honeyflow needs to be well designed for imitating the regular network flow and involves false information to confuse the adversaries. Therefore, we use GAN model which is applied to learn from real hosts' pattern and guide honeypots' communications. Our SDN controller learns the hosts' network behavior and instructs the honeypots to imitate real servers and clients.

To train the GAN model, we consider to use the CIDDS-001 dataset [92] which includes flow-based packets' attributes. From the dataset, we select features [date first seen, duration, source IP address, destination IP address, bytes, packets]. We can simulate clients' behavior by measuring the total number of packets, bytes, and duration to guide the honey connection. From the CIDDS-001 dataset, we randomly select 10 clients and one server. The clients are from the developer subnet and the server is the File Server in the subnet. For each client, we select all the traffic it communicated with the File Server during one week and generate the same number of honeyflow packets. For the File Server, we select all the packets it sent back to the clients upon requests as real data, and generate a honeyflow in the same pattern. As a GAN can process only continuous input attributes, this is a major challenge since flow-based network data consists of categori-

cal attributes (e.g., *IP addresses or port numbers*). Another challenge is how to interpret continuous output vectors to flow-based network data. Therefore, we conceived network dataflow attributes as analogous to **utterances** in natural language. We adopt deep learning approaches for natural language generation tasks in this work to overcome the above challenges. Donahue et al. [93] proposed LaTextGAN to utilize an AutoEncoder to learn a low-dimensional representation of sentences and generate sentences with the improved Wasserstein GAN (WGAN) [94]. We adopt the AutoEncoder (AE) and Long-Short Term Memory (LSTM) model for representing the attributes of the packets. Moreover, we use the AE–LSTM model to encode attributes of network data (e.g., *IP addresses and port number*) into smooth representations. The generator network then trained to generate its own flow-based representations in the learned latent space. Each network flow-based data vector produced by the generator is then passed through the decoder, which decodes to the nearest network attributes.

The generator network produces additional points in this latent variable space, which decode to valid packets. Typically in GAN, the discriminator network is trained to classify real and generated packets from their latent representations. In this work, the generator is the defender that attempts to fool the discriminator by generating realistic packets. On the other hand, the discriminator plays the role of the attacker's passive reconnaissance processes. Our game model assumes that the attacker utilizes the discriminator detection probability as belief.

7.4 Game Model

Let a computer network $\aleph$ may have many hosts or resources which are connected each other. Each node η may contain various types of vulnerabilities. For simplicity, we assume that each node contains only one vulnerability of a type $\tau \in T$, where T is the set of all possible types of vulnerabilities. Here, we consider that the defender uses a deception technique to expose the types of fake vulnerabilities in the network by using honeyflows.

The attacker does not know which node information is real and which one is a honeypot by using packet analysis. We assume the attacker cannot reliably distinguish real and honey flow, but he/she has belief distributions over real and honey flows. These beliefs mainly depend on how realistically the defender generates honey flows, which the defender uses as signals to the attacker. We also assume that the attacker is simply relying on the signals to make his decisions. Therefore, in network $\aleph$, a node $\eta^{\tau} \in \aleph$ can be a real host or a honeypot from the attacker's perspective.

We now formally define the strategies and utilities of the players using the notation listed in Table 7.1.

Table 7.1: Game Notation

$\aleph$	Network
T	set of all types of vulnerabilities in the network
R^{τ}	Number of real nodes containing τ type vulnerability where R is a vector
χ^{τ}	Defender's mixed strategy over each type $\tau \in T$
β	Upper bound vector of fake vulnerabilities indexed by each type of vulnerability $\tau \in T$
ζ^{τ}	Cost of creating each fake τ type of vulnerability using honey flow
P_r^{τ}	Attacker's belief on τ type real flows
P_h^{τ}	Attacker's belief on τ type honeyflows
υ^{τ}	Vulnerability value that is a cost of attacking τ type node
$\psi_{\eta^{\tau}}^{r}$	Importance of a real node η that contains τ type vulnerability
$\psi_{\eta^{\tau}}^{h}$	Importance of a fake node η that contains τ type vulnerability
$\alpha_{\eta^{\tau}}$	Action of attack the node η^{τ} where α_0 is the no-attack action, yielding 0 payoff

Our proposed game model is a Stackelberg two-player normal form non-zero sum game between the defender and attacker. The defender plays a mixed strategy over each type $\tau \in T$ vulnerability, and the attacker plays a pure strategy. Here, we define χ_β^τ as action of the defender to generate maximum $\beta \in \mathbb{N}$ fake τ type vulnerabilities in the network using honey flow that makes the game finite.Therefore, the defender's action space for a particular τ is a set $\{\chi_0^\tau, \ldots, \chi_\beta^\tau\}$ and the defender plays a mixed strategy over it. The defender's overall strategy is to distribute marginal probabilities over all types of vulnerabilities T, which is denoted by χ. The attacker's pure strategy α_{η^τ} represents choosing to attack a node $\eta^\tau \in \aleph$ that contains type τ vulnerability, or not to attack. The notation R^τ denotes the total number of real nodes that contain τ type vulnerability. Our consideration is that the attacker knows the total number of real nodes R^τ but can not distinguish between real and fake nodes.

7.4.1 Utility Functions

The utilities for the players depend on attacker's beliefs, the node the attacker attacks, the importance of that node, the defender's strategy of generating fake vulnerabilities, honey flow generation cost, and the vulnerability value. To reduce calculation complexity, we assume that nodes with the same type of vulnerability have equal importance values. When the attacker targets a real node, he gets a positive reward that is equal to $\psi_{\eta^\tau}^r$, but the reward is negative when the target node is fake where the reward is equal to $\psi_{\eta^\tau}^h$. The rewards are opposite for the defender. In this game, the υ^τ denotes the amount of time and resources the attacker needs to expense to compromise a particular node that contains τ type vulnerability. Also, ζ^τ denotes the defender's cost of exposing each vulnerability which may include honey flow generation cost, network congestion, deploying deceptive host (e.g. honeypot). The notations P_r^τ and P_h^τ are used to represent the attacker's beliefs over real and honey flows respectively for τ type.

Defender's Utility If the defender plays the strategy χ and the attacker's strategy is α_{η}^{τ}, the defender's expected utility is defined as follows:

$$\begin{aligned} U^d(\chi,\eta^\tau) &= \mathcal{F}(\chi,\tau)\left[(P_r^\tau \times (-\psi_{\eta^\tau}^r)) + ((1-P_r^\tau)\times(+\psi_{\eta^\tau}^h))\right] \\ &+ (1-\mathcal{F}(\chi,\tau))\left[(P_h^\tau \times (-\psi_{\eta^\tau}^r)) + (1-P_h^\tau)\times(+\psi_{\eta^\tau}^h))\right] \\ &- \sum_{\tau\in T}\sum_{x\in[0,\beta^\tau]}[\chi_x^\tau \times x \times \zeta^\tau] \end{aligned} \tag{7.1}$$

Here, $\mathcal{F}(\chi,\tau)$ denotes the cumulative function that considers all possible numbers up to the upper bound of creating the fake $\tau \in T$, which can be calculated as follows:

$$\mathcal{F}(\chi,\tau) = \sum_{x\in[0,\beta^\tau]} \chi_x^\tau \left[\frac{R^\tau}{R^\tau + x}\right]$$

Attacker's Utility Analogously, when υ^τ is the attacking cost of a τ type vulnerable node for the attacker, the expected utility is given by:

$$\begin{aligned} U^a(\chi,\eta^\tau) &= \mathcal{F}(\chi,\tau)\left[(P_r^\tau \times (+\psi_{\eta^\tau}^r)) + ((1-P_r^\tau)\times(-\psi_{\eta^\tau}^h))\right] \\ &+ (1-\mathcal{F}(\chi,\tau))\left[(P_h^\tau \times (+\psi_{\eta^\tau}^r)) + (1-P_h^\tau)\times(-\psi_{\eta^\tau}^h))\right] - \upsilon^\tau \end{aligned} \tag{7.2}$$

A Small Example : Consider a network with two categories of vulnerable nodes. Let the attacker's beliefs be $P^r = (1.0, 0.5)$ and $P^h = (0.5, 0.5)$. Let the importance of nodes be $\psi^r = (5, 10)$, and $\psi^h = (5, 5)$ and the attacking costs $\upsilon = (2, 1)$. Let the cost of creating honey flow indicating each type of vulnerability is $\zeta = (1, 0.5)$. Let the total number real flows indicating each type of real vulnerability be $R = (2, 2)$, and the upper bound on honeyflows be $\beta = (2, 2)$. Now, consider if the defender plays the following strategy χ:

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{cc} x^1 & x^2 \end{array} \\ \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \end{array}$$

In the above matrix, we represent a type τ by its index position (e.g., χ^1), and the row indexes are the count of honey vulnerability for each τ. Now, the attacker's best response is to attack vulnerability type-2 with expected utility $U^a(\chi, 2) = 1.5$, and the defender's utility is $U^d(\chi, 2) = -3.75$ when the defender strategy is to create 1 type-1 and 2 type-2 fake vulnerabilities.

7.4.2 Solution Approach

We present a solution method to solve above non-zero sum game. We formulate the normal form Stackelberg formulas using Linear Programming (LP). By solving this program, we can determine the optimal strategy for the defender in the game that generates different numbers of fake vulnerable nodes' by using honey flow. This solution distributes marginal probability over each type of vulnerability $\tau \in T$. The following LP computes the defender's optimal mixed strategy for each type of vulnerability under the constraint that the attacker plays a pure-strategy best response:

$$\max_{\eta^\tau \in \aleph} \; U^d(\chi, \eta^\tau)\, \alpha_{\eta^\tau} \tag{7.3}$$

s.t.

$$U^a(\chi, \eta^\tau)\, \alpha_{\eta^\tau} \geq U^a(\chi, \eta^{\tau^*})\, \alpha_{\eta^\tau} \quad \forall\, \eta^\tau, \eta^{\tau^*} \in \aleph \tag{7.4}$$

$$\chi_x^\tau \in [0,1] \quad \forall \tau \in T \quad \forall x \in [0, \beta^\tau] \quad \forall x, \beta^\tau \in \mathbb{N} \tag{7.5}$$

$$\sum_{x \in [0, \beta^\tau]} \chi_x^\tau = 1 \quad \forall \tau \in T \quad \forall x, \beta^\tau \in \mathbb{N} \tag{7.6}$$

$$\alpha_{\eta^\tau} \in \{0,1\} \quad \eta^\tau \in \aleph \tag{7.7}$$

$$\sum_{\eta^\tau \in \aleph} \alpha_{\eta^\tau} = 1 \tag{7.8}$$

Equation (7.3) represents the objective function reflecting the Stackelberg equilibrium and maximizing the defender's expected utility. In a Strong Stackelberg Game (SSG), the

leader (defender) moves first, and the follower (attacker) observes the leader's strategy and attacks a particular target. The inequality in Equation (7.4) ensures that the attacker plays the best response. A tie means multiple targets provide the same expected utility for the attacker which breaks in favor of the defender. Equation (7.5), created a real variable for each defender's pure strategy χ_x^τ, the action of creating $x \in [0, \beta^\tau]$ fake vulnerabilities for type τ type where β^τ is the upper bound. Therefore, $(\chi_0^\tau, \ldots, \chi_\beta^\tau)$ are unknown variables and the possible pure strategies for the defender for $\tau \in T$. Equation (7.6) forces the defender strategy to be a valid probability distribution over τ and distributes marginal probability over all types. Finally, Equations (7.7–7.8) make sure that the attacker plays a pure strategy best response in the game where α_η^τ is a binary variable.

Our proposed solution can not scale a large size game since the game grows exponentially. We could improve this solution by using advanced methods based on an incremental strategy generation approach. However, our main focus is to analyze the honeyflow deception, not to improve the game-theoretic algorithm. The basic solution is sufficient to solve a mid-size network that successfully meet our goals.

7.5 Game-Theoretic Experiments

In this section, we focus on evaluating the solution quality of the proposed Stackelberg game model. Figure (7.1) investigates the impact of real and honeyflow beliefs in decision making of the attacker with five types of vulnerabilities. We set the number of real vulnerability for each type to 100, and the upper bound on the number of fake vulnerability for each type to 100. We also set the honeyflow generation cost to 0.0001 and the attacking cost zero. Then, we set the same importance for real and fake nodes as 1 and fixed a specific real flow belief while changing the value of honeyflow belief. We observe that the attacker's utilities start increasing when the honeyflow belief is greater than the real flow belief. When the honeyflow belief is higher, the attacker gets high informative information from the honeyflows. In this case, the expected utility of the attacker depends on the

information revealed from the real flow. If the real flow belief is higher than honeyflow, the attacker can identify the real flow with a higher probability once these probabilities become equal, it is more confusing for the attacker to identify the real and fake hosts correctly.

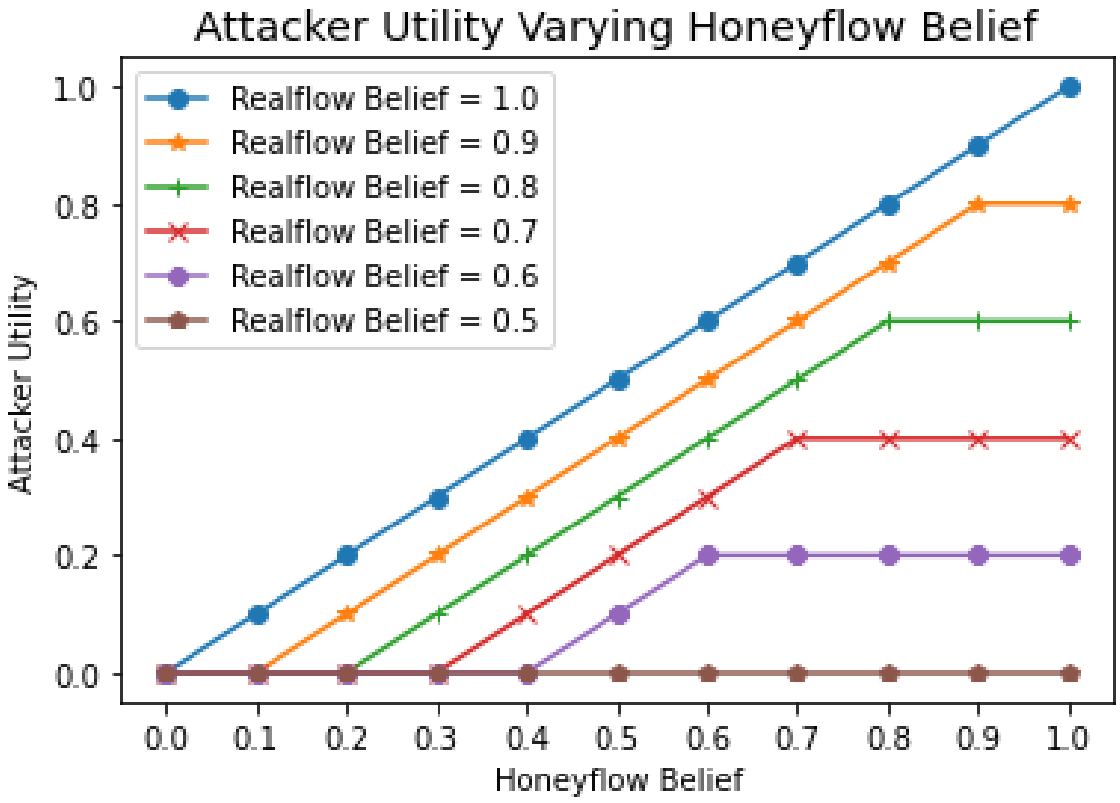


Figure 7.1: Comparison of attacker utility with different vulnerability value

In the following experiment, we investigate the impact of attack cost in the attacker's utility. First, we set the number of real vulnerabilities and the upper bound on the number of fake vulnerabilities to 100. Then, we set the importance of a real host to 1.0 and of a honey host to 0.5. We set real and honeyflow signal probabilities to 0.5. We set the honeyflow generation cost to 0.0001 and consider varying attack costs. The results in Figure (7.2) show that the attacker utility does not change in the game-theoretic solution when the signal of the real and honey are identical. We note that the small attack cost does

not impact the overall result. However, the game-theoretic solution provides lower utility to the attacker than the two-baseline model.

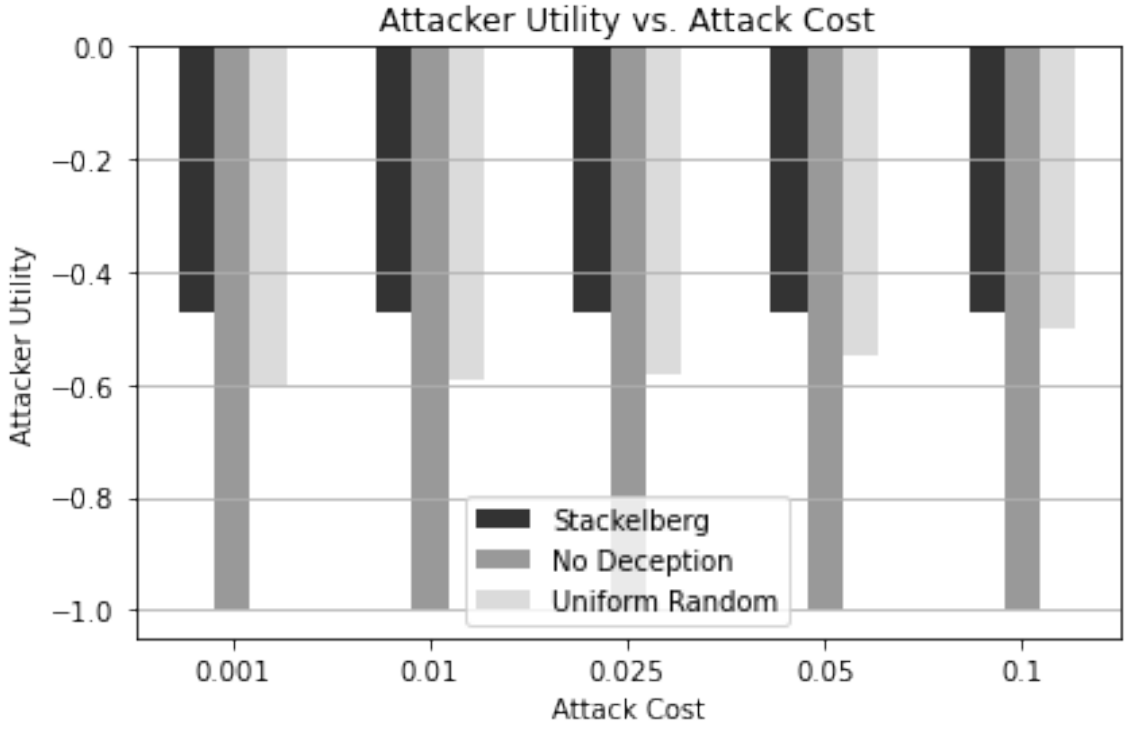


Figure 7.2: Comparison of attacker utility with different vulnerability value

In Figure (7.3), we show the defender's expected utility as we increase the ratio of honeyflows to real flows. In this experiment, we set the number of real vulnerabilities for each type and the upper limit of fake vulnerabilities to 100. We also set the real and fake host importance to 1.0 and 0.5, respectively, as well as the real and honeyflow signal probability to 1 . In the graph, each line represents a different number of real flows in the original game. We see that the defender utility increases as we add honeyflows, but only up to a point; when the marginal value is less than the cost, the optimal solution is to stop adding additional flows. We see this in the shape of the curves. We note that the point where these curves flatten out is similar across all of the different numbers of real flows.

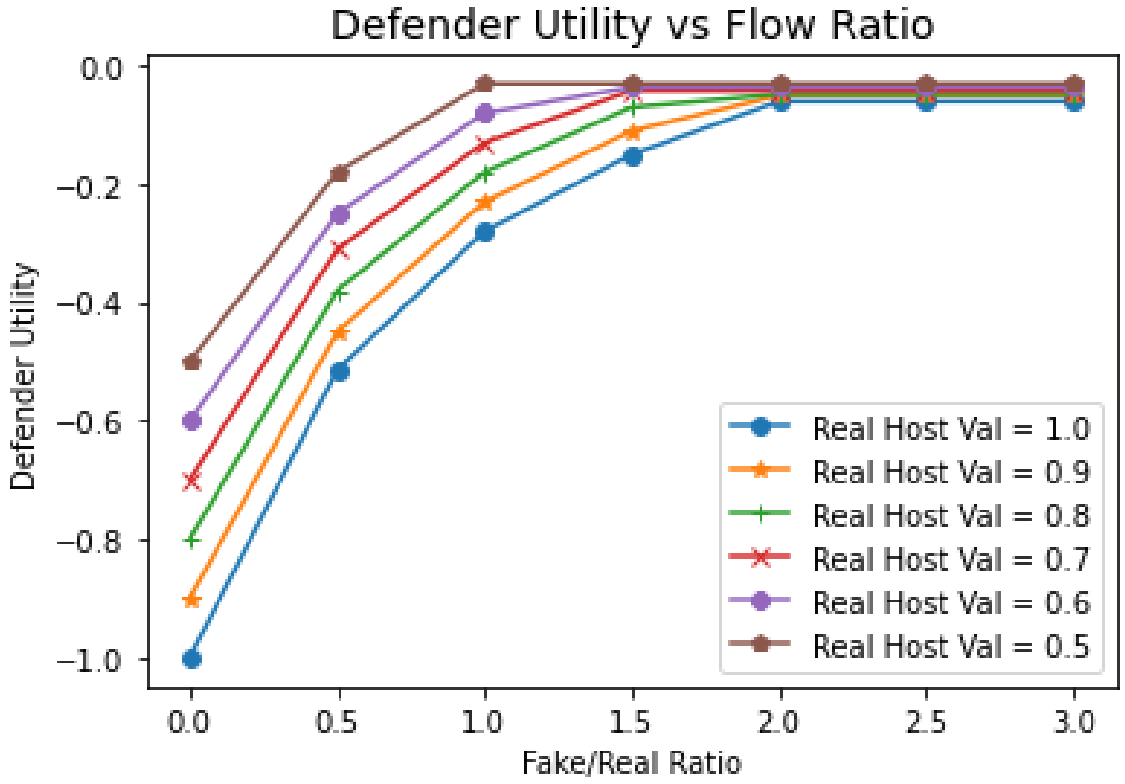


Figure 7.3: Comparison of defender's utility when the honey and real flow ratio is variable

We now analyze how the honeyflow cost impacts the optimal solutions. In Figure (7.4), the network setup consists of five types of vulnerabilities where we set real vulnerabilities in each type and the limit of fake vulnerabilities to 100. We also set real host important value 1.0 and fake host importance 0.5. The real and honey signal probability are 1.0 and 0.5, respectively. Also, the vulnerability value is 0.5. The results in show that the game-theoretic solution significantly outperforms the two baselines in most settings, demonstrating the value of optimizing the honeyflow generation based on the specific scenario. We also note that the cost has a significant impact on the overall result; with a high cost, the game-theoretic solution is similar to not generating flows. Thus, random honey flow generation can be detrimental for the defender. With a low cost, the performance of the game-theoretic solution is identical to the uniform random policy; since flows are so

cheap, it is practical to create a very large number of them without much regard to strategy. With intermediate costs, the value of the strategic optimization is highest, which is the most likely scenario in real applications.

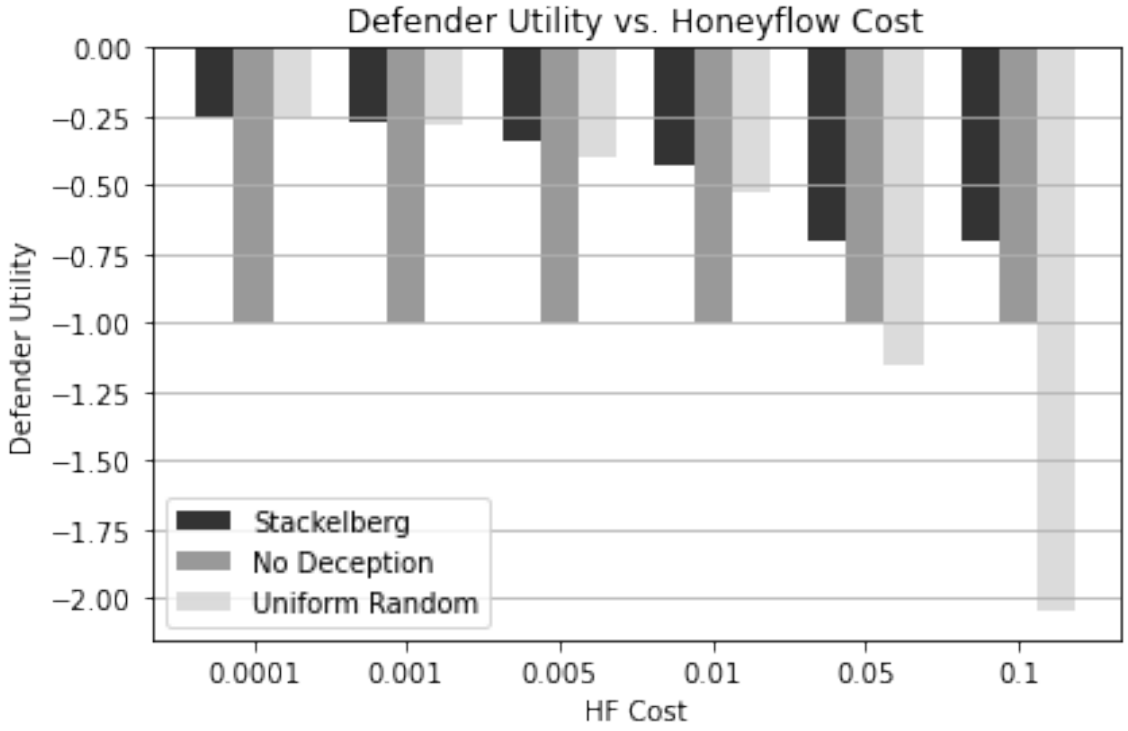


Figure 7.4: Comparison of defender utility when honey flow generation cost is different.

7.6 Conclusion

Network attackers often use traffic analysis as passive reconnaissance to understand network topology and uncover network vulnerabilities. They can interact with a network environment in several stages and stay to conduct an APT attack for an extended time. In this work, we consider a deceptive framework that generates honey flows to carry fake information of various vulnerabilities in a network. We propose a two-player non-zero-sum Stackelberg game that determines the optimal defender's strategy to deploy honey traffic in the presence of an attacker who perceives different beliefs in different types of real

and honeyflows. This model can be used in a dynamic network to balance cost and value trade-offs. The empirical analysis shows that when honeyflows are highly similar to real flows, distinguishing real and honey flows is harder for attackers. We can investigate the qualities of game-theoretic solutions in real-world network for further exploration. Also, this model could be extended where the attackers can update their beliefs based on the traffic analysis.

Chapter 8

Conclusion

8.1 Thesis Summary

This thesis presented several game-theoretical and adversarial machine learning solutions to practice deception further effectively and resolve some limitations of existing formal models that provide little guidance on deception effectiveness in designing and allocating decoys. In chapter 4, we propose Honeypot Feature Selection Game (HFSG) to analyze how to make deceptive objects look real and how real objects can be modified to look more like misleading ones that potentially make the computer network more uncertain to the adversaries. We showed two-sided deception, as we named this approach, is more effective in many situations over one-sided deception where the defender is restricted to making a change in real objects. Our HFSG model gives a new and more nuanced way to think about the quality of different deception strategies and how robust they are to an adversary being able to see through the deception. We can identify which features the defender should focus on modifying to make the deception more effective, including real objects' features. We can also correctly identify cases where deception is not the best solution because the costs of creating a believable deception may be higher than the value they create. However, one limitation of our current HFSG model is that we analyzed the two-sided deception approach using a small number of features. For further exploring and scaling the two-sided deception approach, in chapter 5, we introduce the Two-Sided Generative Adversarial Network (TS-GAN) algorithm that can solve two-sided feature deception on a large and complex basis and provide an approximate optimal solution. We model the problem as a two-player minimax game between the defender and attacker.

The defender attempts to confuse the attacker by creating fake and adversarial samples, the latter of which is created by modifying a real sample in such a way that the attacker mispredicts it as a fake sample. The attacker's goal is to correctly distinguish between real and fake models to minimize his expected loss. On the other hand, the defender's strategy maximizes the attacker's expected loss. Our method can generate fake samples look likes as real samples and real samples looks like as fake samples when the feature space complex an large. In a dynamic learning environment, the technique can also be used as a robust classifier for the binary classification problem.

Our study on deception using fake network traffic to invalidate the attacker's gathering information through network reconnaissance is presented in Chapter 6. Here, we use a normal form game-theoretical model to optimize the honey (fake) flow deployment for a defender. Our results showed that we could apply this model to optimize honey flows in realistic-sized networks with a fast response rate. But for further optimization, we need the scalability that could be improved significantly beyond this basic algorithm. However, this model does not consider the impact of honeyflow quality on attackers' decision-making strategies. Therefore, in Chapter 7, we present a new honeyflow game model to incorporate the attacker's beliefs across different types of honey traffic, reflecting the quality of the honey flows and indicating how difficult it is for the attacker to discriminate. The experimental results show that the quality of honey flow has a substantial impact on an attacker's decision-making method. To effectively mislead an attacker, the deceptive architecture must generate a substantially similar honeyflow to real traffic.

Overall, in this thesis, the HFSG model answers how and when a defender should create decoy objects, including honeypots, honey traffic, and other domains. Then, my TS-GAN algorithm provides a new way of solving the two-sided feature deception problems when the feature spaces are large and complex. Finally, my honeyflow game models characterize how defenders can optimize decoys or honey traffic deployment in an adversary's presence while taking into account a variety of real-world computer network conditions.

8.2 Future Direction

In TS-GAN model, the Generator generates data that looks like the training set, and the Obfuscator manipulates training samples that look like fake data. The architecture of the Generator is similar to the traditional GAN model, where it learns to map a particular distribution to a different distribution, potentially resulting in fake output samples. The existing TS-GAN algorithm could be useful for making decoy objects with more complex and large features. But the existing model can be extended differently using two data sets for real and fake inputs. Instead of mapping a specific distribution to fake data directly, the fake data set will be used for training and manipulated to create real-like fake samples. In this case, the Generator architecture will be similar to the Obfuscator. However, one key challenge is to train the three networks simultaneously. If the real and fake data sets are very different, only manipulating samples' features can lead the training process to the failure or collapse mode. Therefore, this model needs to explore a number of ways, such as employing different types of data sets instead of only images, such as network traffic, malware, and many other forms of data.

An additional extension I want to incorporate is feature modification constraint loss, which imposes an extra cost for modifying some particular features.The penalty will only be applied when the model adds perturbation more than a particular threshold. In the current version of TS-GAN, the Obfuscator model uses two loss functions, with the distance loss penalizing more when feature manipulation is considerably large. On the other hand, the obfuscation loss aims to create a perturbed vector that encourages the real sample is missclassified as a fake class. The introduction of modification constraint loss must pose significant challenges, especially in selecting the non-modification feature group and drawing the relationship between the features and their costs. But it will help to understand the two-deception problem's theoretical qualities better.

References

[1] Lance Spitzner, *Honeypots: tracking hackers*, vol. 1, Addison-Wesley Boston, 2002.

[2] Christopher Kiekintveld, Viliam Lisy, and Radek Pibil, "Game-theoretic foundations for the strategic use of honeypots in network security," *Advances in Information Security*, vol. 56, pp. 81–101, 2015.

[3] Stefan Achleitner, Thomas La Porta, Patrick McDaniel, Shridatt Sugrim, Srikanth V Krishnamurthy, and Ritu Chadha, "Cyber deception: Virtual networks to defend insider reconnaissance," in *Proceedings of the 8th ACM CCS international workshop on managing insider security threats*. ACM, 2016, pp. 57–68.

[4] Nandan Garg and Daniel Grosu, "Deception in honeynets: A game-theoretic analysis," in *2007 IEEE SMC Information Assurance and Security Workshop*. IEEE, 2007, pp. 107–113.

[5] Ma Yanli Zhao Zhansheng Huang Xuan, "Honeypot-network trap [j]," *Computer Engineering and Applications*, vol. 4, 2003.

[6] Thomas E Carroll and Daniel Grosu, "A game theoretic investigation of deception in network security," *Security and Communication Networks*, vol. 4, no. 10, pp. 1162–1172, 2011.

[7] Palvi Aggarwal, Cleotilde Gonzalez, and Varun Dutt, "Looking from the hacker's perspective: Role of deceptive strategies in cyber security," in *2016 International Conference On Cyber Situational Awareness, Data Analytics And Assessment (CyberSA)*. IEEE, 2016, pp. 1–6.

[8] Jun Zhuang, Vicki M Bier, and Oguzhan Alagoz, "Modeling secrecy and deception in a multiple-period attacker–defender signaling game," *European Journal of Operational Research*, vol. 203, no. 2, pp. 409–418, 2010.

[9] Christopher Kiekintveld, Viliam Lisỳ, and Radek Píbil, "Game-theoretic foundations for the strategic use of honeypots in network security," in *Cyber Warfare*, pp. 81–101. Springer, 2015.

[10] Zheyuan Ryan Shi, Ariel D Procaccia, Kevin S Chan, Sridhar Venkatesan, Noam Ben-Asher, Nandi O Leslie, Charles Kamhoua, and Fei Fang, "Learning and planning in feature deception games," *arXiv preprint arXiv:1905.04833*, 2019.

[11] Radek Píbil, Viliam Lisý, Christopher Kiekintveld, Branislav Bošanský, and Michal Pěchouček, "Game Theoretic Model of Strategic Honeypot Selection in Computer Networks," , no. 1, pp. 201–220, 2012.

[12] Aaron Schlenker, Omkar Thakoor, Haifeng Xu, Fei Fang, Milind Tambe, Long Tran-Thanh, Phebe Vayanos, and Yevgeniy Vorobeychik, "Deceiving cyber adversaries: A game theoretic approach," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2018, pp. 892–900.

[13] Aaron Schlenker, Omkar Thakoor, Haifeng Xu, Fei Fang, Milind Tambe, and Phebe Vayanos, "Game theoretic cyber deception to foil adversarial network reconnaissance," in *Adaptive Autonomous Secure Cyber Systems*, pp. 183–204. Springer, 2020.

[14] Omkar Thakoor, Milind Tambe, Phebe Vayanos, Haifeng Xu, Christopher Kiekintveld, and Fei Fang, "Cyber camouflage games for strategic deception," in *International Conference on Decision and Game Theory for Security*. Springer, 2019, pp. 525–541.

[15] Kevin P Dyer, Scott E Coull, and Thomas Shrimpton, "Marionette: A programmable network traffic obfuscation system," in *24th {USENIX} Security Symposium ({USENIX} Security 15)*, 2015, pp. 367–382.

[16] A. J. Pinheiro, J. M. Bezerra, and D. R. Campelo, "Packet padding for improving privacy in consumer iot," in *2018 IEEE Symposium on Computers and Communications (ISCC)*, 2018, pp. 00925–00929.

[17] Yong Guan, Xinwen Fu, Dong Xuan, Prashanth Umesh Shenoy, Riccardo Bettati, and Wei Zhao, "Netcamo: camouflaging network traffic for qos-guaranteed mission critical applications," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 31, no. 4, pp. 253–265, 2001.

[18] Ertugrul Ciftcioglu, Rommie Hardy, Kevin Chan, Lisa Scott, Diego Oliveira, and Gunjan Verma, "Chaff allocation and performance for network traffic obfuscation," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018, pp. 1565–1568.

[19] Ertugrul N Ciftcioglu, Rommie L Hardy, Lisa M Scott, and Kevin S Chan, "Efficient chaff-aided obfuscation in resource constrained environments," in *MILCOM 2017-2017 IEEE Military Communications Conference (MILCOM)*. IEEE, 2017, pp. 97–102.

[20] Kevin Leyton-Brown and Yoav Shoham, "Essentials of game theory: A concise multidisciplinary introduction," *Synthesis lectures on artificial intelligence and machine learning*, vol. 2, no. 1, pp. 1–88, 2008.

[21] Jiarui Gan and Bo An, "Minimum support size of the defender's strong stackelberg equilibrium strategies in security games," in *Proc. AAAI Spring Symp. on Appl. Computat. Game Theory*, 2014.

[22] George Leitmann, "On generalized stackelberg strategies," *Journal of optimization theory and applications*, vol. 26, no. 4, pp. 637–643, 1978.

[23] Michele Breton, Abderrahmane Alj, and Alain Haurie, "Sequential stackelberg equilibria in two-person games," *Journal of Optimization Theory and Applications*, vol. 59, no. 1, pp. 71–97, 1988.

[24] Mandiant Intelligence Center, "Apt1: Exposing one of china's cyber espionage units," *Mandiant, Tech. Rep*, 2013, `https://www.fireeye.com/content/dam/fireeye-www/services/pdfs/mandiant-apt1-report.pdf`.

[25] Nikos Virvilis, Bart Vanautgaerden, and Oscar Serrano Serrano, "Changing the game: The art of deceiving sophisticated attackers," in *2014 6th International Conference On Cyber Conflict (CyCon 2014)*. IEEE, 2014, pp. 87–97.

[26] Cliff Changchun Zou and Ryan Cunningham, "Honeypot-aware advanced botnet construction and maintenance," in *International Conference on Dependable Systems and Networks (DSN'06)*. IEEE, 2006, pp. 199–208.

[27] Neal Krawetz, "Anti-honeypot technology," *IEEE Security & Privacy*, vol. 2, no. 1, pp. 76–79, 2004.

[28] Cliff Stoll, *The cuckoo's egg: tracking a spy through the maze of computer espionage*, Doubleday, 1989.

[29] Abhishek Mairh, Debabrat Barik, Kanchan Verma, and Debasish Jena, "Honeypot in network security: a survey," in *Proceedings of the 2011 international conference on communication, computing & security*. ACM, 2011, pp. 600–605.

[30] Marcin Nawrocki, Matthias Wählisch, Thomas C Schmidt, Christian Keil, and Jochen Schönfelder, "A survey on honeypot software and data analysis," *arXiv preprint arXiv:1608.06249*, 2016.

[31] Matthew L Bringer, Christopher A Chelmecki, and Hiroshi Fujinoki, "A survey: Recent advances and future trends in honeypot research," *International Journal of Computer Network and Information Security*, vol. 4, no. 10, pp. 63, 2012.

[32] Niels Provos, "Honeyd-a virtual honeypot daemon," in *10th DFN-CERT Workshop, Hamburg, Germany*, 2003, vol. 2, p. 4.

[33] Radek Píbil, Viliam Lisỳ, Christopher Kiekintveld, Branislav Bošanskỳ, and Michal Pěchouček, "Game theoretic model of strategic honeypot selection in computer networks," in *International Conference on Decision and Game Theory for Security*. Springer, 2012, pp. 201–220.

[34] T. Alpcan and T. Başar, *Network security: A decision and game-theoretic approach*, Cambridge University Press, 2010.

[35] A. Laszka, Y. Vorobeychik, and X. D. Koutsoukos, "Optimal personalized filtering against spear-phishing attacks.," in *AAAI*, 2015.

[36] E. Serra, S. Jajodia, A. Pugliese, A. Rullo, and VS Subrahmanian, "Pareto-optimal adversarial defense of enterprise systems," *ACM Transactions on Information and System Security (TISSEC)*, vol. 17, no. 3, pp. 11, 2015.

[37] A. Schlenker, H. Xu, M. Guirguis, C. Kiekintveld, A. Sinha, M. Tambe, S. Sonya, D. Balderas, and N. Dunstatter, "Don't bury your head in warnings: A game-theoretic approach for intelligent allocation of cyber-security alerts," in *IJCAI*, 2017.

[38] A. Schlenker, O. Thakoor, H. Xu, F. Fang, M. Tambe, L. Tran-Thanh, P. Vayanos, and Y. Vorobeychik, "Deceiving cyber adversaries: A game theoretic approach," in *AAMAS*, 2018.

[39] W. Wang and B. Zeng, "A two-stage deception game for network defense," in *Decision and Game Theory for Security*, 2018.

[40] Kun Wang, Miao Du, Sabita Maharjan, and Yanfei Sun, "Strategic honeypot game model for distributed denial of service attacks in the smart grid," *IEEE Transactions on Smart Grid*, vol. 8, no. 5, pp. 2474–2482, 2017.

[41] Quang Duy La, Tony QS Quek, Jemin Lee, Shi Jin, and Hongbo Zhu, "Deceptive attack and defense game in honeypot-enabled networks for the internet of things," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 1025–1035, 2016.

[42] Miao Du, Yongzhong Li, Qing Lu, and Kun Wang, "Bayesian game based pseudo honeypot model in social networks," in *International Conference on Cloud Computing and Security*. Springer, 2017, pp. 62–71.

[43] Neil C Rowe, E John Custy, and Binh T Duong, "Defending cyberspace with fake honeypots," *JOURNAL OF COMPUTERS*, vol. 2, no. 2, pp. 25, 2007.

[44] Leyi Shi, Junnan Zhao, Lanlan Jiang, Wenjuan Xing, Jian Gong, and Xin Liu, "Game theoretic simulation on the mimicry honeypot," *Wuhan University Journal of Natural Sciences*, vol. 21, no. 1, pp. 69–74, 2016.

[45] Fabio De Gaspari, Sushil Jajodia, Luigi V Mancini, and Agostino Panico, "Ahead: A new architecture for active defense," in *Proceedings of the 2016 ACM Workshop on Automated Decision Making for Active Cyber Defense*. ACM, 2016, pp. 11–16.

[46] Christian Kroer and Tuomas Sandholm, "Extensive-form game abstraction with bounds," in *Proceedings of the fifteenth ACM conference on Economics and computation*. ACM, 2014, pp. 621–638.

[47] Gaspard Monge, "Mémoire sur la théorie des déblais et des remblais," *Histoire de l'Académie Royale des Sciences de Paris*, 1781.

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

[49] Ling Huang, Anthony D Joseph, Blaine Nelson, Benjamin IP Rubinstein, and JD Tygar, "Adversarial machine learning," in *Proceedings of the 4th ACM workshop on Security and artificial intelligence*. ACM, 2011, pp. 43–58.

[50] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[51] Ian Goodfellow, "Nips 2016 tutorial: Generative adversarial networks," *arXiv preprint arXiv:1701.00160*, 2016.

[52] Palvi Aggarwal, Yinuo Du, Kuldeep Singh, and Cleotilde Gonzalez, "Decoys in cybersecurity: An exploratory study to test the effectiveness of 2-sided deception," *arXiv preprint arXiv:2108.11037*, 2021.

[53] Mohammad Sujan Miah, Marcus Gutierrez, Oscar Veliz, Omkar Thakoor, and Christopher Kiekintveld, "Concealing cyber-decoys using two-sided feature deception games," in *Proceedings of the 53rd Hawaii International Conference on System Sciences*, 2020.

[54] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

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

[56] Ian Goodfellow, Nicolas Papernot, Patrick McDaniel, Reuben Feinman, Fartash Faghri, Alexander Matyasko, Karen Hambardzumyan, Yi-Lin Juang, Alexey Kurakin, Ryan Sheatsley, et al., "cleverhans v0. 1: an adversarial machine learning library," *arXiv preprint arXiv:1610.00768*, vol. 1, 2016.

[57] Charles V Wright, Scott E Coull, and Fabian Monrose, "Traffic morphing: An efficient defense against statistical traffic analysis.," in *NDSS*. Citeseer, 2009, vol. 9.

[58] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow, "Transferability in machine learning: from phenomena to black-box attacks using adversarial samples," *arXiv preprint arXiv:1605.07277*, 2016.

[59] Warren He, James Wei, Xinyun Chen, Nicholas Carlini, and Dawn Song, "Adversarial example defense: Ensembles of weak defenses are not strong," in *11th {USENIX} workshop on offensive technologies ({WOOT} 17)*, 2017.

[60] Nicholas Carlini and David Wagner, "Towards evaluating the robustness of neural networks," in *2017 ieee symposium on security and privacy (sp)*. IEEE, 2017, pp. 39–57.

[61] Yann LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[62] Md Mosharaf Hossain and Mohammad Sujan Miah, "Evaluation of different svm kernels for predicting customer churn," in *2015 18th International Conference on Computer and Information Technology (ICCIT)*. IEEE, 2015, pp. 1–4.

[63] Jon Latimer, *Deception in War: Art bluff value deceit most thrilling episodes cunning mil hist from the trojan*, Abrams, 2003.

[64] Sushil Jajodia, V Subrahmanian, Vipin Swarup, and Cliff Wang, *Cyber deception*, vol. 6, Springer, 2016.

[65] Karel Horák, Quanyan Zhu, and Branislav Bošanskỳ, "Manipulating adversary's belief: A dynamic game approach to deception by design for proactive network security," in *International Conference on Decision and Game Theory for Security*. Springer, 2017, pp. 273–294.

[66] Palvi Aggarwal, Omkar Thakoor, Aditya Mate, Milind Tambe, Edward A Cranford, Christian Lebiere, and Cleotilde Gonzalez, "An exploratory study of a masking strategy of cyberdeception using cybervan," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*. SAGE Publications Sage CA: Los Angeles, CA, 2020, vol. 64, pp. 446–450.

[67] Lance Spitzner, "Honeypots: Catching the insider threat," in *19th Annual Computer Security Applications Conference, 2003. Proceedings.* IEEE, 2003, pp. 170–179.

[68] Quan Hoang, Tu Dinh Nguyen, Trung Le, and Dinh Phung, "Mgan: Training generative adversarial nets with multiple generators," in *International conference on learning representations*, 2018.

[69] Arnab Ghosh, Viveka Kulharia, Vinay P Namboodiri, Philip HS Torr, and Puneet K Dokania, "Multi-agent diverse generative adversarial networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8513–8521.

[70] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[71] Weiwei Hu and Ying Tan, "Generating adversarial malware examples for black-box attacks based on gan," *arXiv preprint arXiv:1702.05983*, 2017.

[72] Alonso Granados, Mohammad Sujan Miah, Anthony Ortiz, and Christopher Kiekintveld, "A realistic approach for network traffic obfuscation using adversarial machine learning," in *International Conference on Decision and Game Theory for Security*. Springer, 2020, pp. 45–57.

[73] Chaowei Xiao, Bo Li, Jun-Yan Zhu, Warren He, Mingyan Liu, and Dawn Song, "Generating adversarial examples with adversarial networks," *arXiv preprint arXiv:1801.02610*, 2018.

[74] Group-IB and Fox-IT, "Anunak: Apt against financial institutions," `https://www.group-ib.com/resources/threat-research/Anunak_APT_against_financial_institutions.pdf`, 2014.

[75] Tim Matthews, "Operation Aurora – 2010's major breach by Chinese hackers," `https://www.exabeam.com/information-security/operation-aurora/`, 2019.

[76] Tabu S. Kondo and Leonard J. Mselle, "Penetration testing with banner grabbers and packet sniffers," *Journal of Emerging Trends in computing and information sciences*, vol. 5, no. 4, 2014.

[77] Genevieve Bartlett, John Heidemann, and Christos Papadopoulos, "Understanding passive and active service discovery," in *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement (IMC)*. 2007, pp. 57–70, ACM.

[78] Dan Levin, Marco Canini, Stefan Schmid, Fabian Schaffert, and Anja Feldmann, "Panopticon: Reaping the benefits of incremental SDN deployment in enterprise networks," in *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, Philadelphia, PA, 2014, pp. 333–345, USENIX Association.

[79] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

[80] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013.

[81] Markku Antikainen, Tuomas Aura, and Mfikko Särelä, "Spook in your network: Attacking an sdn with a compromised openflow switch," in *Secure IT Systems*, Karin Bernsmed and Simone Fischer-Hübner, Eds., Cham, 2014, pp. 229–244, Springer International Publishing.

[82] Kevin Benton, L. Jean Camp, and Chris Small, "Openflow vulnerability assessment," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, New York, NY, USA, 2013, HotSDN '13, pp. 151–152, ACM.

[83] Samuel Jero, Xiangyu Bu, Cristina Nita-Rotaru, Hamed Okhravi, Richard Skowyra, and Sonia Fahmy, "BEADS: Automated attack discovery in OpenFlow-based SDN systems," in *Proceedings of the International Symposium on Research in Attacks, Intrusions, and Defenses (RAID)*, 2017, vol. 10453 of *LNCS*, pp. 311–333.

[84] Jeffrey Pawlick, Edward Colbert, and Quanyan Zhu, "A game-theoretic taxonomy and survey of defensive deception for cybersecurity and privacy," *ACM Computing Surveys (CSUR)*, vol. 52, no. 4, pp. 82, 2019.

[85] Gérard Wagener, Alexandre Dulaunoy, Thomas Engel, et al., "Self adaptive high interaction honeypots driven by game theory," in *Symposium on Self-Stabilizing Systems*. Springer, 2009, pp. 741–755.

[86] Yue Yin, Bo An, Yevgeniy Vorobeychik, and Jun Zhuang, "Optimal deceptive strategies in security games: A preliminary study," in *Proc. of AAAI*, 2013.

[87] Bo An, Milind Tambe, Fernando Ordonez, Eric Shieh, and Christopher Kiekintveld, "Refinement of strong stackelberg equilibria in security games," in *Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

[88] Omkar Thakoor, Milind Tambe, Phebe Vayanos, Haifeng Xu, and Christopher Kiekintveld, "General-sum cyber deception games under partial attacker valuation information.," in *AAMAS*, 2019, pp. 2215–2217.

[89] Xiaotao Feng, Zizhan Zheng, Prasant Mohapatra, and Derya Cansever, "A stackelberg game and markov modeling of moving target defense," in *International Conference on Decision and Game Theory for Security*. Springer, 2017, pp. 315–335.

[90] Iffat Anjum, Mohammad Sujan Miah, Mu Zhu, Nazia Sharmin, Christopher Kiekintveld, William Enck, and Munindar P Singh, "Optimizing vulnerability-driven honey traffic using game theory," *arXiv preprint arXiv:2002.09069*, 2020.

[91] Milind Tambe, *Security and game theory: algorithms, deployed systems, lessons learned*, Cambridge university press, 2011.

[92] Markus Ring, Daniel Schlör, Dieter Landes, and Andreas Hotho, "Flow-based network traffic generation using generative adversarial networks," *Computers & Security*, vol. 82, pp. 156–172, 2019.

[93] David Donahue and Anna Rumshisky, "Adversarial text generation without reinforcement learning," *ArXiv*, vol. abs/1810.06640, 2018.

[94] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville, "Improved training of Wasserstein GANs," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2017, NIPS'17, p. 5769–5779, Curran Associates Inc.

Vita

Mohammad Sujan Miah is a Ph.D. student in Computer Science at the University of Texas at El Paso. Currently, he is working at the Intelligent Agents and Strategic Reasoning Lab under Dr. Christopher Kiekintveld. His research interest lies between game theory and adversarial machine learning areas. His main research focuses on the application of game theory in cyber defense. Particularly, he has been exploring deceptive strategies using the game-theoretic model. He received a Bachelor of Science in Computer Science and Engineering from the University of Dhaka, Bangladesh. After graduation, he worked as a software engineer at a Samsung Electronic research center for a few years. He also had professional experience working as a Sr. software engineer at other leading software company of Bangladesh.