The *What, When,* and *How* of Strategic Movement in Adversarial Settings: A Syncretic View of AI and Security

by

Sailik Sengupta

A Dissertation Presented in Partial Fulfillment
of the Requirement for the Degree
Doctor of Philosophy

Approved September 2020 by the
Graduate Supervisory Committee:

Subbarao Kambhampati, Chair
Tiffany (Youzhi) Bao
Dijiang Huang
Guoliang Xue

ARIZONA STATE UNIVERSITY

December 2020

ABSTRACT

The field of cyber-defenses has played catch-up in the cat-and-mouse game of finding vulnerabilities followed by the invention of patches to defend against them. With the complexity and scale of modern-day software, it is difficult to ensure that all known vulnerabilities are patched; moreover, the attacker, with reconnaissance on their side, will eventually discover and leverage them. To take away the attacker's inherent advantage of reconnaissance, researchers have proposed the notion of proactive defenses such as Moving Target Defense (MTD) in cyber-security. In this thesis, I make three key contributions that help to improve the effectiveness of MTD.

First, I argue that naïve movement strategies for MTD systems, designed based on intuition, are detrimental to both security and performance. To answer the question of *how to move*, I (1) model MTD as a leader-follower game and formally characterize the notion of *optimal movement strategies*, (2) leverage expert-curated public data and formal representation methods used in cyber-security to obtain parameters of the game, and (3) propose optimization methods to infer strategies at Strong Stackelberg Equilibrium, addressing issues pertaining to scalability and switching costs. Second, when one cannot readily obtain the parameters of the game-theoretic model but can interact with a system, I propose a novel multi-agent reinforcement learning approach that finds the optimal movement strategy. Third, I investigate the novel use of MTD in three domains– cyber-deception, machine learning, and critical infrastructure networks. I show that the question of *what to move* poses non-trivial challenges in these domains. To address them, I propose methods for patch-set selection in the deployment of honey-patches, characterize the notion *differential immunity* in deep neural networks, and develop optimization problems that guarantee differential immunity for dynamic sensor placement in power-networks.

To all those who inspired me to do a doctorate,

but will never read this thesis.

ACKNOWLEDGEMENTS

While I wasn't confident in my choice to do a doctorate at the time I applied, I am happy I did. I am even happier that I had Prof. Subbarao Kambhampati (or endearingly, Rao) as my mentor in graduate school. His ability to grasp abstruse concepts almost instantly and draw profound connections between seemingly unrelated subjects left me in complete awe after I joined the lab. Further, his debate skills, along with his inherent flamboyance, made the task of convincing him that an idea is technically sound, novel, and worth pursuing an onerous one. Yet somehow, over all these years, he taught me how to become a researcher, perfectly balancing between guiding me and letting me develop skills to think and write independently. Beyond research, his talent in coming up with clever titles (and patience to edit paper-drafts) right before submission deadlines, devotion to teaching, open-mindedness to discuss any topic under the sun, and an innate sense of justice have imparted lessons and inculcated in me several virtues that (I believe) have made me a better person.

I have been taught by and collaborated with some of the best faculties in the department. Prof. Xue's optimization course was both theoretically rigorous and encouraged me to add it as a tool to our research toolbox. As the thesis will show, I have tried to judiciously use this powerful tool. I am immensely grateful to Prof. Huang and Prof. Doupe for giving me their time and the opportunity to discuss research ideas. They have taught me the fundamentals of cloud network and software security, often patiently explaining what modeling aspects escapes the realm of pragmatism and correcting my paper drafts. My biggest regret remains the lack of collaboration with Prof. Bao, who also subscribes to the philosophy that intelligent and strategic thinking can greatly improve the effectiveness of cyber-defenses (I wish our paths cross in the future). I have deeply admired and learned from Prof. Lee's rigor at formulating a problem statement, and Prof. Sen's skill at carefully and patiently

dissecting a proposed idea until all latent assumptions lay bare. Lastly, my heartfelt gratitude extends to Monica (who had an answer to all our non-academic questions), Pam, Theresa, and Lee– they all made my time in the department so much easier.

I have come to immensely appreciate the role of collaboration and conversations with my peers is shaping me as a researcher. Gautam and Tathagata have helped me identify flaws in the technical arguments, provided constructive feedback, corrected numerous paper drafts, and, most importantly, helped me look less stupid in front of Rao. With Ankur, I have had the most fruitful collaborations– learning about cloud-network security, discussing and formulating problems from scratch, setting up experiments, and writing papers. Sarath, Anagha, and Lydia (alongside me) have helped raise the average decibel level of Yochan Lab. They have taught me a lot about automated planning by engaging me in numerous technical discussions and correcting my mistakes ranging from misremembering author names to understanding recondite concepts. They have switched roles, acting as a critical reviewer of my ideas, and participated in discussions about life, relationships, books, coffee, food, and whatnot. It amazes me how, alongside their own immensely successful graduate careers, they managed to do all this. My path with Zahra, Alberto, and Niharika crossed because of our mutual interests in game theory and machine learning– topics that are relatively unpopular in the lab; I wish they had joined earlier! I also had a chance to work with Abdulhakim, Marthony, Andrew, Aditya, Daniel, Soumya, Garima, and Karthik during their time at ASU, and am thankful to Gabe, Yantian, Sriram, Utkarsh, Lin, and Mudit for making my years in Yochan memorable. I will cherish the chats with Kartik, J, Biplav, and Minh at multiple conferences; with all the stories about research and gossip from yesteryears, they have made my Yochan experience complete.

I am grateful to IBM for the prestigious PhD Fellowship award. My heartfelt thanks to Ian & J.R. Rao for inviting me to present at the IBM-MIT AI-security

workshop and IBM T.J. Watson Center, to Fred & Jiyong for collaborating with me in uniting the two most popular proactive defenses in cyber-sec. My gratitude extends to my collaborators at Amazon research– Rashmi, He, Batool, Spandana, and Mona– who introduced me to the field of Natural Language Processing and helped me grow over a period of two internships; to Prof. Amit Konar & Prof. Atal Chaudhuri (at JU), to Prof. Sangamitra Bandyopadhyay & Debarka Da (at ISI, Kolkata), and Soham who all inculcated in me the love for research during my undergraduate years. Finally, a note of thanks to Prof. Tambe and Prof. Vorobeychik for providing valuable insights, during conference-chats and doctoral consortium discussions, and encouraging me to work on many of the problems that have made it into this thesis.

Thanking Sachin and Kaustav just as a research collaborator would appear remiss. They have spent countless hours dealing with my complaints about research, the proverbial reviewer #2, life and its challenges. Having dealt with my idiosyncrasies, I am thankful Kaustav has been kind enough not to kick me out of our residence (even after proofreading my thesis). I consider myself lucky to have had a long list of friends who have always been there for me, some a call or text away and some a knock-on-the-door afar– Soumik, Sankarshan, Sudipa, Madhumita, Saket, Aditi, Rohit, Akash, Angad, Bahar, Sandipan, Arindam, Anisha, Parijat, Jayeeta, Kallol & Monosrija, Sayahnika, Deepasmita, Sriloy, Yugarshi, Abhik, the Xaverian gang, and the list goes on.

At last, without my family, I would have never made it this far. To all those who believed in me and supported me– Mom, Dad, Sis, Mashi-Mesho, my aunts & uncles, my in-laws, and my brilliant set of cousins (who inspired me but made me look like an underachiever). Only a sentence to thank my dearest friend and my wife, Sri, seems unfair; yet, here it goes. She has made me a better person, a better planner, a more diligent researcher, a more compassionate friend, . . . ; to her, I owe my life.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

CHAPTER Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Figure Page

Chapter 1

INTRODUCTION

Strategy without [movement] is the slowest route to victory.

[Movement] without strategy is the noise before defeat.

– Sun Tzu (The Art of War)

The complexity and scale of modern-day cyber-systems make cyber-defense a challenging problem. While research in uncovering novel vulnerabilities are of utmost importance, the system administrators land up playing catch-up in this cat-and-mouse chase of vulnerability discovery and patch design. Moreover, the need for on-time deployments and the scarcity of security skills among the majority of system designers and developers result in the deployment of systems that are easy targets for attackers.

Most of the modern defense mechanisms, while necessary, are insufficient to thwart cyber-adversaries. First, an attacker, with time on their side, can spend reconnaissance effort in modeling the attack surface and then carefully plan their attacks. Second, implementation of the defenses in practice is often far from ideal, at times introducing new attack surfaces and thereby more opportunities for an attacker to exploit the system. Experts have predicted that by the end of 2020, 99% of the vulnerabilities exploited by adversaries will be known to security and IT professionals since a year ago [3]. This is not surprising given the time and complexity associated with routine maintenance and security patching of current production systems. Third, *zero-day* attacks developed by an attacker, leveraging information during the reconnaissance phase, render traditional defenses ineffective.

To address these challenges inherent in static cyber-defenses, the paradigm of proactive defenses has emerged as a promising solution in the cyber-security community. The objective of these defenses is to take away an adversary's inherent asymmetric advantage of reconnaissance by continually moving aspects of the cyber system. While methods like Cyber Deception [4] try to deceive an attacker by providing them with incorrect information about the system, Moving Target Defense (MTD) [5] introduces randomness in the behavior of a system, rendering adversary's information-gathering actions nugatory at attack time. These defenses not only seek to level the playing field for the defender but also act as a defense-in-depth strategy that can be leveraged alongside static defense mechanisms to provide better security. This thesis characterizes MTDs under a formal umbrella that helps us to (1) increase the effectiveness of existing MTDs by improving movement strategies, and (2) introduce MTD as a novel defense-in-depth solution in several application domains, identifying key challenges associated with the defender's strategy-set selection and developing methods to address them.

In the first part of the thesis, we propose new and adapt existing artificial intelligence (AI) techniques in multi-agent systems for improving the movement strategies of existing MTDs. Note that random movement is a key aspect in the success of MTD systems, as predictable changes can be easily exploited by the adversary. In this regard, existing works merely consider naïve movement strategies, such as unbiased randomization, to move between the various system configurations of an MTD, completely ignoring the adversary's capability of behaving strategically. Even if we consider known attacks, such strategies can prove to be highly sub-optimal with weak security guarantees and heavy performance impacts. To answer the question of *how to move*, we first model the attacker-defender interaction in an MTD scenario as a two-player game. Then, we can either (1) leverage existing knowledge to infer or (2)

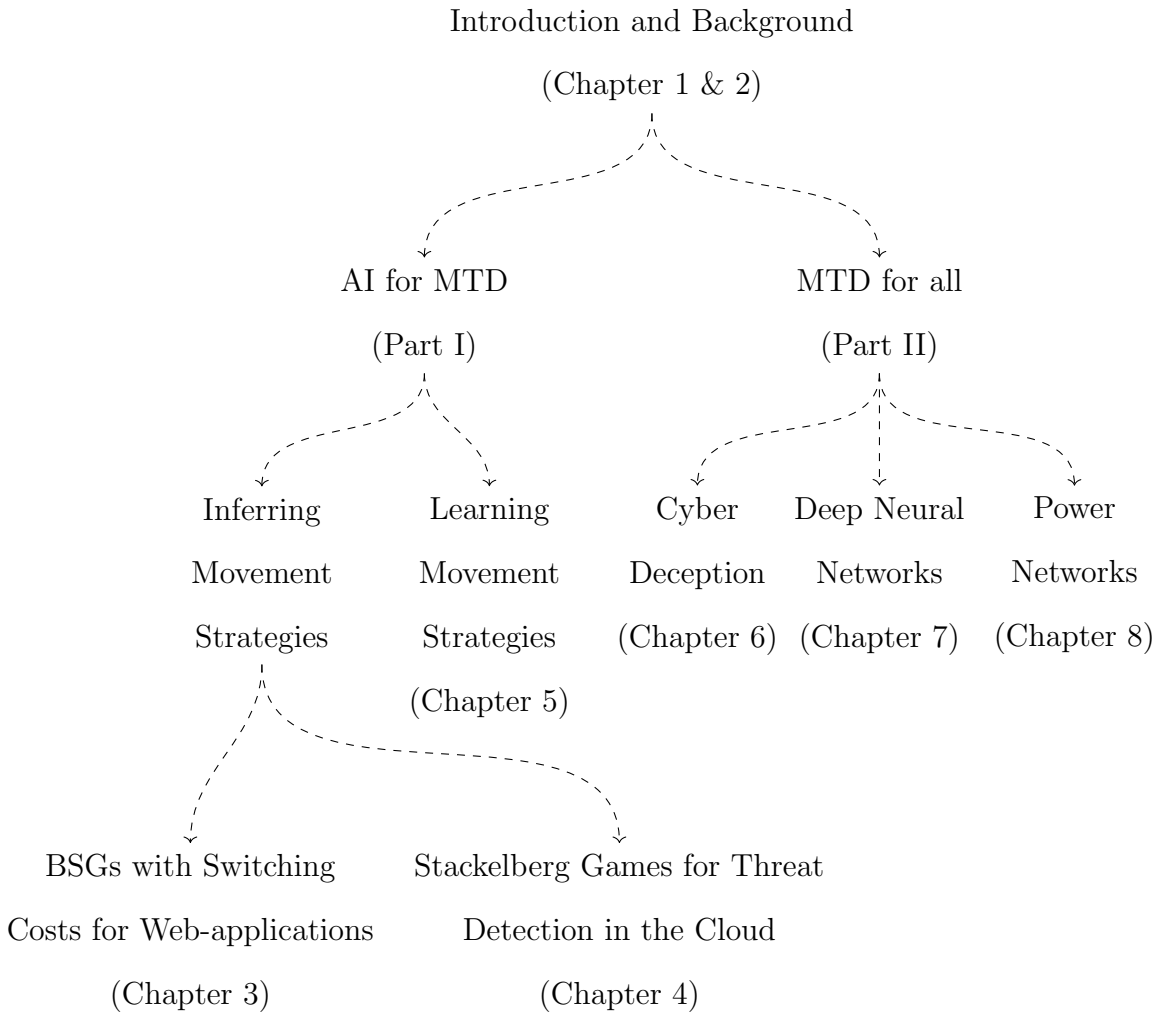


Figure 1.1: An overview of the thesis.

interact to learn optimal movement strategies that achieve the desired balance between performance and security impacts of MTDs. This part of the thesis labeled as *AI for MTD* in Figure 1.1, our proposed inference and learning methods address challenges arsing out of (1) non-uniform switching costs, (2) large action sets of players, and (3) multi-stage attacks in cloud systems. We show that both the inferred and the learned strategies are at equilibrium and outperform state-of-the-art strategies proposed in the security community.

In the second part of the thesis, we consider the novel use of Moving Target Defense for multiple application domains. Specifically, we consider the use of MTD in three domains: cyber-deception, image classification, and sensor placement in critical infrastructure networks. In all the three domains, we consider different aspects of *what to move.* In the context of cyber-deception, we highlight the challenge of conflict-free patch set selection and propose an end-to-end framework that can be leveraged for the deployment of honey-patches alongside regular software patches to boost software security. In the context of machine learning and critical infrastructure networks, we identify the key challenge of ensuring *differential immunity*, i.e. the degree to which an attack can cripple all the configurations of an MTD. We propose a metric to formalize the notion of adversarial attack transferability [6] for an ensemble of Deep Neural Networks, and show it is indicative of how effective an MTD will be. At last, for unique detection of failure points in power networks, we generalize existing graph-theoretic formalism to guarantee complete differential immunity. Our generalization can be extended to an array of graph-theoretic solution concepts, enabling the effective use of MTD in other domains in the future.

Most of the chapters discussed in this thesis consider the notion of *model-based strategy inference* where we first leverage existing domain knowledge to construct a game-theoretic model of the interaction and between the attacker and the defender and then propose methods to find strategies at equilibrium. In contrast, the work in Chapter 5 considers a *model-free strategy learning* approach where via repeated interaction we simply learn the strategy at equilibrium. An intermediate approach would be to learn the model and then leverage the model-based inference mechanisms to find strategies. Such an approach, although not discussed in this thesis, may be desirable in settings where the learned parameters of a game can be leveraged to bootstrap the model that needs to be constructed for similar domains.

1.1 Thesis Overview

The thesis, as shown in Figure 1.1, is divided into nine chapters. We provide a brief overview of the chapters to follow.

(Ch. 2) We introduce the reader to works in Moving Target Defense (MTD) and a set of concepts in game-theory that are leveraged throughout the document. A list of useful notations is enlisted at the end of this chapter.

(Ch. 3) To formally characterize the problem of finding a good movement strategy, we model the interaction between an MTD system and an attacker as a Bayesian Stackelberg Game (BSG) and obtain the game parameters by leveraging public attack data curated by security experts. In the context of software systems in general and web-applications in particular, we consider the costs of switching actions and show that the single-stage normal-from BSGs cannot account for it, resulting in movement strategies that are too costly to deploy in the real-world. We propose optimization techniques to infer movement strategies that try to balance between achieving high security and reducing switching costs.

(Ch. 4) Cyber adversaries often leverage attack plans and exploit multiple vulnerabilities to meet specific attack goals while defenders consider the deployment of Intrusion Detection Systems (IDS) to detect such attack actions. The deployment of all possible IDSs can severely impact the Quality of Service (QoS) and thus, we consider a moving target approach to shifting a subset of detection mechanisms. First, we model the MTD scenario initially as a Stackelberg Security Game (SSG) and impose a novel utility structure; the utility values are obtained using (1) exploit information in publicly-accessible databases and (2) existing formalism in network security. This modeling assumes the attacker is

capable of exploiting any vulnerability in the system and detection mechanisms are always accurate when detecting attacks. To relax these assumptions, we then propose a Markov Game approach to find movement strategies that provide higher security and reduce the performance impact. We demonstrate the efficacy of our strategies in small-scale cloud security scenarios.

(Ch. 5) When the parameters of the game that models an MTD system cannot be readily obtained but interaction with a simulator or real-world cyber system is possible, we consider if a model-free approach to learn movement policies. We first propose a novel game-theoretic framework that trades-off between the generality offered in modeling multiple attacker types and the sample complexity of learning. We then propose an approach, similar to approaches in multi-agent Reinforcement Learning, to learn movement strategies at Stackelberg equilibrium.

(Ch. 6) We introduce the dynamic aspect of MTD in cyber-deception. Specifically, we consider the strategic deployment of honey-patches in production systems. While we can leverage the methods developed in the earlier chapters to determine optimal movement strategies in these settings, developing a defender's action set for real-world deployment poses a key challenge. To address this problem, we propose the use of existing, albeit less popular, version-control systems in developing *conflict-free* defender actions and propose and end-to-end system architecture that facilitates patch-set deployments.

(Ch. 7) To increase the robustness of deep neural networks, we introduce MTDeep– an MTD approach for randomization at test-time. We define the notion of differential immunity that provides a formal metric to gauge the transferability of attacks against an ensemble; higher differential immunity implies higher effectiveness of MTDeep. For norm-based perturbation attacks, we show

that by formulating the interaction between the ensemble and the users as a Bayesian Stackelberg Game, we can optimize for both accuracy and robustness. Further, MTD is the first defense-in-depth solution that can be applied in conjunction with existing defenses and can help to improve the security guarantees afforded by these defenses.

(Ch. 8) In certain domains, constructing a defender's action set $A^{\mathcal{D}}$ that is differentially immune may be possible. In this regard, we show that by leveraging MTD for the placement of power measurement units to monitor anomalies in high-voltage transformers, one can robustify failure identification in power networks. Existing use of graph-theoretic modeling in this domain helps us design methods that guarantee differentially immune MTD configurations and, thus, provide better security.

(Ch. 9) We conclude the thesis by (1) showing how the works presented achieve the goals of this thesis, (2) reflect on various aspects of the presented work, and (3) highlight key takeaways.

Chapter 2

BACKGROUND

Table of Contents ◇ 1 ◇ 2 ◇ 3 ◇ 4 ◇ 5 ◇ 6 ◇ 7 ◇ 8 ◇ 9

In this chapter, we formally define Moving Target Defense and leverage the categorization proposed in [7] to situate the works presented in this thesis. We then introduce some preliminaries in game-theory that will be helpful to understand the upcoming chapters. At the end, we provide a list of notations that appear throughout the thesis (and can thus be used for quick reference).

2.1 Moving Target Defense

A Moving Target Defense (MTD) can be defined using the tuple $\langle C, t, M \rangle$ where $C = \{c_1, c_2, \ldots c_n\}$ denotes the set of system configurations that an MTD can switch between, t denotes the timing function that determines the time at which the next move action occurs, and $M : H \rightarrow C$ denotes the movement function where H is the history of the system configurations it has switched to in the past. [1] In most cases, we will assume H is either $\emptyset$ or Markovian and t is a constant function, i.e. the movement occurs after a fixed-time period.

The three variables that define an MTD can help us characterize a particular defense based on how it answers the following questions– (1) what to move (C), (2) when to move (t) and (3) how to move (M). In this section, we will categorize the answers to the questions and identify the membership of several MTDs proposed in

[1] As an MTD seeks to make the attacker uncertain about its success at the time of the attack, M needs to be a random function to provide security benefits.

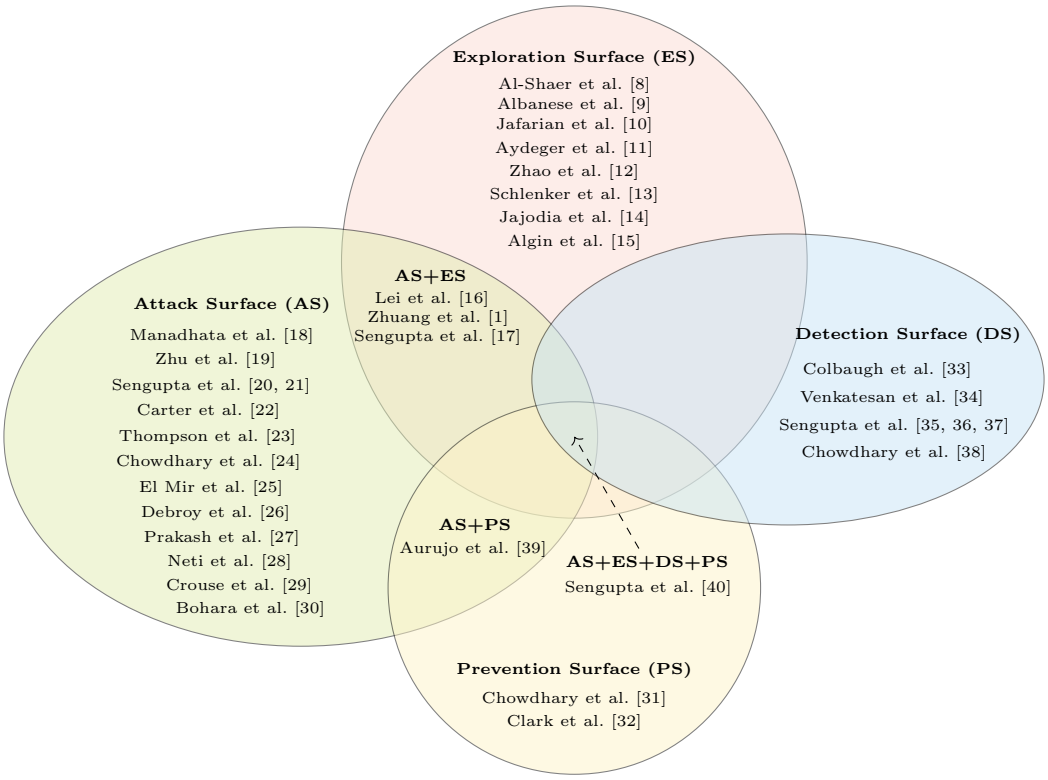


Figure 2.1: The various system surfaces that can be moved by a Moving Target Defense (MTD). Movement of the exploration surface makes it more difficult for an attacker to figure out the exact configuration of the underlying system. Moving the attack surface makes an attack, designed based on reconnaissance, ineffective at attack time. Moving the detection surface, similar to 'patrolling methods' in physical system security, helps in providing efficient detection in budget-constrained (in cyber and cyber-physical) systems. Movement of the prevention surface makes certain stages of an attack such as data ex-filtration difficult. Several works exist at the intersection of the various areas; in these cases, the movement provides benefits associated with shifting two or more surfaces.

the literature to these categories. This categorization helps us describe the landscape of existing work and situate our contributions. Further, it provides (1) a common terminology to describe any MTD and (2) a quick insight into the assumptions that are made by them. An elaborate discussion in the context of MTDs for cyber-security can be found in our survey paper [7].

2.1.1 The Configuration Set C – What to Move?

The components of a system that are of importance to an adversary can be broadly categorized into four surfaces– the exploration surface, the attack surface, the detection surface, and the prevention surface. As described in Figure 2.1, the goal of moving the different surfaces differ.

Our works on inferring movement for MTDs in web-application security move the attack surface [41, 21]. The goal is to ensure that the reconnaissance information, collected over time, used by the adversary to craft attacks, becomes stale at attack time because the system shifts to a new configuration. In regards to shifting the detection surface, we consider works in cloud-network security [35, 42, 38] and the security of fingerprinting failures in power-networks [37]. The goal of these works is to reduce the performance impact/cost of placing all possible detection sensors while ensuring the objectives of sensor placement– attack detection for cloud-network security and unique identification of failures in power-networks– are achieved with high probability.

While the majority of existing works in MTD shift a single surface, the idea of developing MTDs that can shift multiple semantic surfaces of a system is a budding research topic (see works at the intersection of the various sets in Figure 2.1). In this regard, our thesis presents three works. In the context of image classification with deep neural networks [43], we shift the classification surface. While this is the

attack surface for adversarial perturbations, it duals up as the exploration surface for model theft and construction of black-box attacks. For cyber-deception, we seek to continuously move the code-set deployed to production [39]. This continually shifts (1) the attack surface as the attacker is unsure which vulnerabilities can be exploited and (2) the prevention surface as the defender can choose to gather exploit information about a chosen set of vulnerabilities in the system. The third work on learning movement strategies for MTD systems [40] can be applied to all MTDs, regardless of the surface they shift; we only expect that repeated interaction with the MTD environment (simulation, emulation or real-world test-bed) is possible. Hence, it falls at the intersection of all the surfaces shown in Figure 2.1.

2.1.2 The Timing Function t – When to Move?

Given the configuration space C, the timing function seeks to answer the question of *when* should a defender move from one $c \in C$ to a $c' \in C$? Existing works in MTD can be broadly classified into two categories– ones that have a constant time-period for switching and others that have variable time-period switching. As shown in Figure 2.2, most of our works [21, 20, 35, 42, 39] consider a constant time-period T provided as a hyper-parameter to the MTD. Often this time period is determined based on empirical studies [23], but it does not explicitly consider the history of movement or leverage information about attack events to modify T.

On the other hand, there exist two sub-categories under variable time-period switching. First, one can specify that a move is triggered at the occurrence of a particular event– while dependent on the type of the event, often such timing functions adhere to Markovian assumptions. Our work on moving the classification surface in Chapter 7 abides by this notion; the event that triggers the move is the classifi-

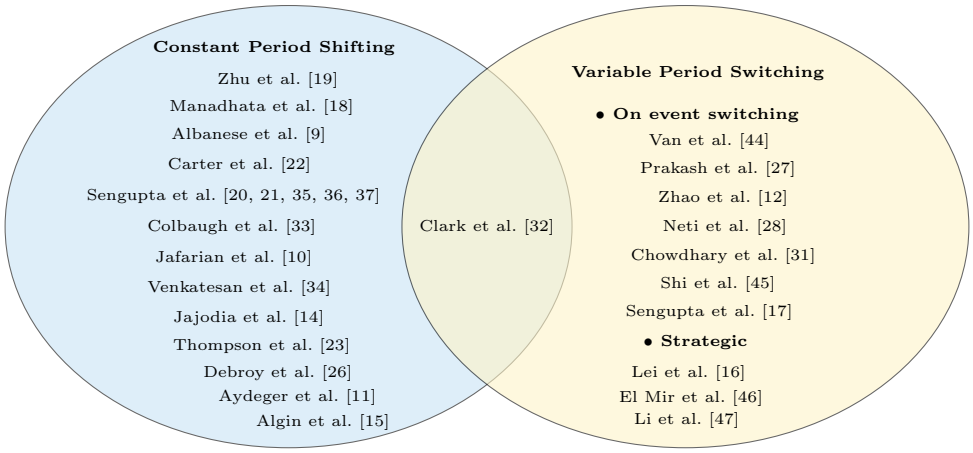


Figure 2.2: The timing function that determines *when* a change occurs in an MTD system can either be (1) a constant function with a pre-defined time-period decided based on some empirical evidence or expert knowledge or (2) vary based on either the occurrence of an event that triggers the move or formally define the notion of an optimal timing function and propose ways to find it.

cation of a new input image. Second, an active line of research exists in developing methods to find a good timing function based on the history of events. Some of these works [16, 47] are inspired by our works on the game-theoretic formulation of MTD [21, 20] and investigate the notion of formally defining and inferring an optimal timing function for MTD for web-applications.

2.1.3 The Movement Function M – How to Move?

As stated above, the movement function $M : H \rightarrow C$ needs to be a random function that takes the history of system configurations as an argument and outputs the configuration that the defender should shift to. In theory, H can incorporate an arbitrarily long trace of events that occurs in an MTD system to decide on the movement strategy. However, this results in scalability issues. Hence, similar to

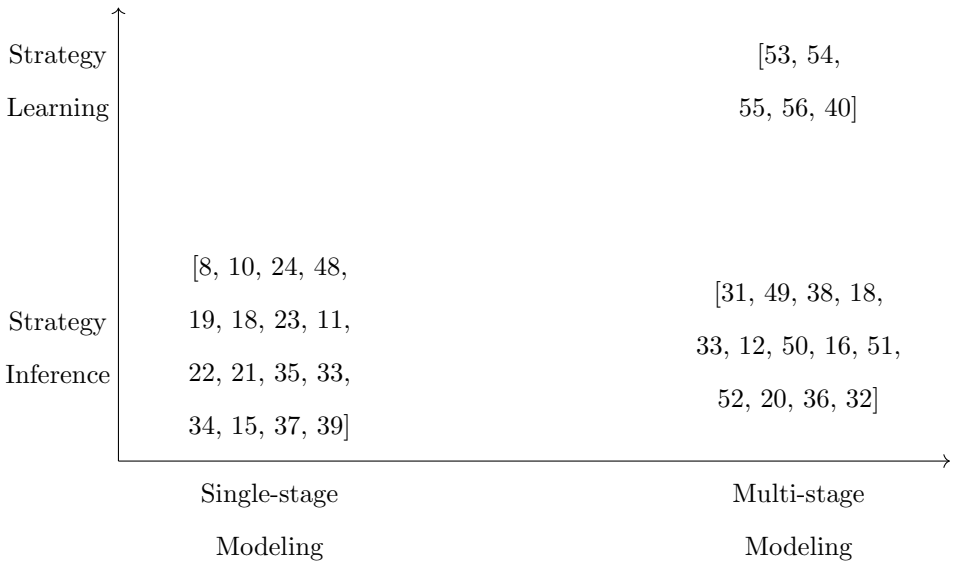


Figure 2.3: Game-theoretic modeling for Moving Target Defenses (MTDs) may either model the interaction as a single-stage or a multi-stage game. This helps us characterize optimal movement policies. Complementary to the modeling aspect, one can propose methods to leverage existing knowledge and infer the optimal movement strategies or learn them via interaction with an environment. The choices made by a particular MTD dictates their position in this space.

existing works, we only consider scenarios where either $H = \emptyset$ or $H = C$. The latter, by definition, adheres to the Markovian assumption. In contrast to works that consider a strategic timing function [47], we do not (need to) consider the time at which a switch occurs as a part of H as we either have a constant-time period or on-event switching.

In scenarios where an MTD is modeled as a single-stage game, $H = \emptyset$ and the goal of the modeling is to come up with a single (mixed) strategy that can be leveraged for movement. Multiple works that are a part of this thesis, such as MTD for web-application security [21, 20], cyber-deception [39], image classification [17] and power-networks [37], fall under this category. On the other hand, defense actions may often

depend on the history or the current state of the system. When $H \neq \emptyset$, the MTD should be modeled as a multi-stage interaction. Our works in cloud network security [35, 42, 38] and web-application security [40] fall under this category.

Another important axis of categorization that emerges, given the works presented in this thesis, is the notion of *inferring vs. learning* movement strategies. While game-theoretic modeling developed in multi-agent systems need to be leveraged regardless of this choice, the approaches to find the movement strategy differ significantly. When inferring a movement strategy, we assume that information required by the game-theoretic modeling can be obtained by leveraging publicly available data or domain-dependent heuristics. Most of the works described in this thesis consider the inference of strategies. On the contrary, when there is no trivial way of obtaining the parameters of the model, but an interaction with an environment is possible, one can, by clever and repeated interaction, learn the optimal movement function. We investigate this line of research in [40].

Note that a major contribution of this thesis is to show that improving M is a key aspect of improving the security and performance of MTDs. In this regard, we consider the use of game-theoretic modeling. Further, we show that existing works that designed M based on intuition, due to lack of formal modeling, are often doomed to be sub-optimal when faced with a strategic adversary. Before discussing our work in the upcoming chapters, we now provide a brief overview of some relevant concepts and terminology in game theory.

2.2 Game Theory

This section intends to set the stage for understanding game-theoretic considered in upcoming chapters. Rather than defining the concepts in the most general sense, we

describe a set of preliminary concepts in the context of two-player games. Further, we will stick to the notion of single-stage games and generalize these terms and definitions to Markov games in the relevant chapters to follow.

We will consider the two players to be the defender $\mathcal{D}$ and the attacker $\mathcal{A}$. The name itself is meant to hint that we will be considering multi-agent settings that are non-cooperative or adversarial in nature. The term *pure-strategy* represents an action that a player in the game can perform. For example, in a game of rock-paper-scissor (RPS), playing rock in a pure strategy. The set of all pure-strategies is called the *Pure-strategy set* or the *Action Set*. We will denote the action set of the players using the notations $A^{\mathcal{A}}$ and $A^{\mathcal{D}}$. In RPS, the cardinality of the action set {rock, paper, scissor} consists of three pure-strategies. A *mixed-strategy* is a probability distribution over the pure-strategy set, i.e. over the actions of the action set. For example, a mixed strategy represented as the probability distribution $\langle 0.2, 0.3, 0.5 \rangle$ for RPS denotes that a player can play rock with a probability of 0.2, paper with a probability of 0.3 and scissor with a probability of 0.5. In this thesis, we will denote the mixed-strategy of the defender using x and the mixed strategy of the attacker using q. Note that the mixed strategy is an ordered vector representing the probability of playing a pure strategy and thus, more aptly, should be represented using the notations $\vec{x}$ (or $\vec{q}$). For convenience, we will drop the vector notation and use x ($/q$) to represent $\vec{x}$ ($/\vec{q}$). To refer to the probabilities of a particular action, say $a \in A^{\mathcal{D}}$ (or $\in A^{\mathcal{A}}$), we use a subscript along with the mixed-strategy variable, i.e. x_a (or q_a).

Given that we have two-players, the reward/utility of each player can be defined using a two-dimensional matrix. For $\mathcal{D}$ and $\mathcal{A}$, these matrices are denoted using $R^{\mathcal{D}}$ and $R^{\mathcal{A}}$. The (i, j)-th entry of the matrix, denoted using $R_{ij}^{\mathcal{A}/\mathcal{D}}$, represents the reward/utility obtained when the player $\mathcal{D}$ (often termed as the *row-player*) plays

	b_1	b_2
a_1	2,3	4,0
a_2	1,1	3,3

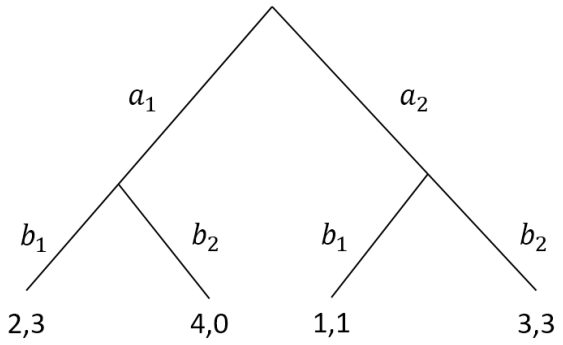


Figure 2.4: The normal and extensive form representation of an example two-player game.

action i and $\mathcal{A}$ (termed as the *column-player*) plays j. While the rewards $R_{ij}^{\mathcal{A}/\mathcal{D}}$ can represent random variables (eg. if rewards are drawn from a distribution) or complicated function (eg. if rewards are a mathematical function of time), we consider them to be real numbers. When the game is a *constant-sum game*, it implies that $R^{\mathcal{A}} + R^{\mathcal{D}} = c$. Thus, the utility of the players can be simply represented using a single reward matrix because other player's utility can be obtained by simply subtracting the reward of the first player from the constant c. In *general-sum games*, a relation between $R^{\mathcal{A}}$ and $R^{\mathcal{D}}$ may not even exist. Most of the work in this thesis considers the latter case; although exploiting structures, when they exist, can help in speeding up inference.

The notion of a good *vs.* bad decision in game-theory is generally represented as some form of equilibrium. In the normal-form game shown (using a bi-matrix table) in Figure 2.4, note that a_1 is a *strictly dominant* strategy for the row-player $\mathcal{D}$, i.e. $\mathcal{D}$ is better off playing a_1 than playing a_2 regardless of what the other player plays. Realizing this, the rational choice for the column player $\mathcal{A}$ is to play b_1, yielding a utility of 2 to $\mathcal{D}$ and 3 to $\mathcal{A}$. Thus, by the concept of *iterated strict dominance*, in which one iteratively removes strictly dominated strategies, (a_1, b_1) is the optimal

strategy in the game. Note that neither of the players has an incentive to *unilaterally deviate* from this play, i.e. if one player sticks to their strategy, the other player cannot obtain a gain in utility by switching their strategy. Thus, this is also the *Nash equilibrium* of the normal form game. In general, a game may allow multiple Nash equilibria; in such cases, the one(s) that gives the highest utility (for a particular player) is called the optimal Nash equilibrium.

Often, one can allow for a notion of commitment in such games [57]. When we allow the row-player $\mathcal{D}$ to commit to a strategy and consider this information is communicated to the other player $\mathcal{A}$ before they play, the notion of Nash Equilibrium does not (always) result in the optimal rewards. For example, in the game shown in Figure 2.4, if $\mathcal{D}$ played a_2 and $\mathcal{A}$ was aware of this before making their move, then $\mathcal{A}$ should play b_2 yielding a reward of 3 to both players. Thus, when this structure of commitment and communication exists, the notion of *Stackelberg Equilibrium* becomes important [58].

The act of committing to a strategy and communicating it to the other player is different from the notion of full observability of actions (which exist in games like Chess, Go, etc.). For example, the player $\mathcal{D}$ might commit to the strategy $\langle 0.1, 0.9 \rangle$ and communicate this strategy to $\mathcal{A}$. With this knowledge, $\mathcal{A}$ does not know the exact action $\mathcal{D}$ will play but can still base their strategy using this extra piece of information. As we will only consider mixed Stackelberg strategies, this distinction is an important one. Now, we discuss two important aspects of Stackelberg Equilibrium strategies– (1) its existence and (2) its solution quality.

Existence of Stackelberg Equilibrium The particular notion of Stackelberg Equilibrium (SE), termed as the Strong Stackelberg Equilibrium (SSE) considered

in the context of Moving Target Defenses (MTD) discussed in the thesis, always exists. The distinction between Strong *vs.* Weak SE arises when given the defender's (in general, the leader's) strategy x, the attacker's (in general, the follower's) best response, denoted by the set $Q(x)$ is not a singleton set. The elements of the set $Q(x)$ often exhibit an interesting property– all $q(\in Q(x))$ yield the same reward to the attacker but yield different reward values for the defender.

A Weak SE (WSE) makes the *pessimistic* assumption that the attacker will choose the response $q \in Q(x)$ that results in the worst-case reward for the defender. Thus, a strategy x^* is WSE for the defender if

$$min_{q \in Q(x)} x^{*T} R^{\mathcal{D}} q = sup_x min_{q \in Q(x)} x^T R^{\mathcal{D}} q = V^{\mathcal{D}}_{wse} \tag{2.1}$$

where $V^{\mathcal{D}}_{wse}$ denotes the Weak Stackelberg Game value for $\mathcal{D}$ when allowing mixed strategies. In such cases, it can be shown (via examples) that the game may not admit a mixed strategy equilibrium, i.e the *supremum* cannot always be replaced by a *maximum* [59, 57]. An interesting aspect is that if one considers only the set of pure strategies for the defender, any finite two-player game is guaranteed to have a weak SE [60] but it may not yield the maximum value $V^{\mathcal{D}}_m$ attainable (i.e. a pure strategy SE might not attain the supremum of Equation 2.1).

On the other hand, a Strong SE (SSE) assumes that the attacker is primarily concerned about maximizing their payoff given a leader's commitment to strategy x and does not mind choosing the strategy $q \in Q(x)$ that favors the defender, i.e. selects the strategy q in the set Q(x) that yields the highest reward to $\mathcal{D}$. Thus, a strategy x^* is SSE for the defender if

$$max_{q \in Q(x)} x^{*T} R^{\mathcal{D}} q = sup_x max_{q \in Q(x)} x^T R^{\mathcal{D}} q = V^{\mathcal{D}}_{sse} \tag{2.2}$$

where $V_{sse}^{\mathcal{D}}$ denotes the Strong Stackelberg Game value for $\mathcal{D}$ when allowing mixed strategies. Note that in this setting the *supremum* can always be replaced with the maximum operator. To understand why, first notice that the set $Q(x)$ always will contain at least one pure strategy in support of the mixed strategy. To prove this, let us consider the optimization problem that characterizes the set $Q(x)$ [61].

$$\begin{aligned} & max_q && \sum_{j \in A^{\mathcal{A}}} \sum_{i \in A^{\mathcal{D}}} R_{i,j}^{\mathcal{A}} x_i q_j \\ s.t. \quad & \sum_{j \in A^{\mathcal{A}}} q_j &= 1 \\ & q_j &\geq 0 \quad \forall j \in A^{\mathcal{A}} \end{aligned} \tag{2.3}$$

In this 'linear' program, the value of x, representing the leader's strategy, is given. Hence, finding the pure strategy j in the attacker's pure strategy set $A^{\mathcal{A}}$ that maximizes $\sum_{i \in A^{\mathcal{D}}} R_{i,j}^{\mathcal{A}} x_i$ should characterize an optimal solution. To look at it from another way, if we break the first constraints into two minimization constraints, the constraint matrix is totally uni-modular and thus, admits integral optimal solutions. Hence, $Q(x)$ has at least one pure-strategy that yields the optimal reward for the attacker.

Given this fact, one can only consider the pure-strategy responses of the attacker to compute the defender's strategy x. For each pure-strategy response $a \in A^{\mathcal{A}}$, the defender can solve the following set of linear (maximization) programs and choose the x^* that results in the highest value [62].

$$\begin{aligned} & max_x && \sum_{i \in A^{\mathcal{D}}} R_{i,a}^{\mathcal{D}} x_i \\ s.t. \quad & \sum_{i \in A^{\mathcal{D}}} R_{i,a}^{\mathcal{A}} x_i &\geq \sum_{i \in A^{\mathcal{D}}} R_{i,a'}^{\mathcal{A}} x_i \quad \forall\ a' \in A^{\mathcal{A}} \\ & \sum_{i \in A^{\mathcal{A}}} x_i &= 1 \\ & x_i &\geq 0 \quad \forall i \in A^{\mathcal{D}} \end{aligned} \tag{2.4}$$

While this linear program may not be feasible for all pure-strategies (especially when the pure strategy is strictly dominated), it will be feasible for at least one pure-strategy (given our previous discussion that any x, at attacker has a pure-strategy best-response). Thus, we can always find an x^*, thereby showing that an SSE (for Equation 2.2) will always exist.

In our works, we formulate MTD as a Bayesian normal-form game in [21, 20, 17, 39] and as a a normal-form game in [35, 37]. In all these scenarios, we adapt either the multiple-LP approach [62] or the single Mixed-Integer Quadratic Programming (MIQP) approach [61]. Thus, the existence guarantees continue to hold in our context. In discounted stochastic games, our zero-sum formulation in [38] always admits a *maximin* Markov strategy (that is equivalent to SE). We show that our general-sum formulations in [63, 40] also admit a Markovian SSE, although sub-optimality may result because of the Markovian assumption [64].

Uniqueness and optimality of SSEs Given the multiple LP approach, discussed in Equation 2.4, it is evident that multiple SSE may exist for a given Stackelberg Game, but regardless of the SSE chosen, they all yield in the same utility for the defender (equal to the SSE game value). For example, there may be two pure-strategies for $\mathcal{A}$, say $a, a' \in A^{\mathcal{A}}$, that yield the same SSE game value for the defender, but correspond to the defender selecting different (pure or mixed) strategies x and x'. While the strategy pairs (a, x) and (a', x') are both SSE strategies, the game value, as determined by Equation 2.2, remains the same. Thus, all strategies at SSE are optimal for the defender.

Table 2.1: Table of Notations

$\mathcal{D}$	$\triangleq$	Defender
$\mathcal{A}$	$\triangleq$	Attacker
$A^{\mathcal{D}}$	$\triangleq$	Pure strategy/actions of the defender
$A^{\mathcal{A}}$	$\triangleq$	Pure strategy/actions of the attacker
$x/\vec{x}$	$\triangleq$	Mixed strategy of the defender
$q/\vec{q}$	$\triangleq$	Mixed strategy of the attacker
x_a/q_a	$\triangleq$	probability of the defender/attacker choosing action $a \in A^{\mathcal{D}}/A^{\mathcal{A}}$
$R^{\mathcal{D}}$	$\triangleq$	Rewards/Utility for the defender
$R^{\mathcal{A}}$	$\triangleq$	Rewards/Utility for the attacker
$R_{ij}^{\mathcal{A}/\mathcal{D}}$	$\triangleq$	Reward/Utility of the attacker/defender when defender play action i and attacker play action j

Part I

Artificial Intelligence for Moving Target Defense (AI for MTD)

Chapter 3

INFERRING MOVEMENT STRATEGIES AT STACKELBERG EQUILIBRIUM IN BAYESIAN GAMES WITH SWITCHING COSTS FOR WEB-APPLICATION SECURITY

Table of Contents ◇ 1 ◇ 2 ◇ 3 ◇ 4 ◇ 5 ◇ 6 ◇ 7 ◇ 8 ◇ 9

C	Attack Surface Shifting in Web-applications
t	Constant/Fixed Time Period
M	Stackelberg Strategy of a Bayesian Normal-form Game with Switching Costs

---

Web-applications, extensively used by an array of businesses, often handle sensitive business and user data. Vulnerabilities present in these web-applications pose a serious threat to the confidentiality, integrity, and availability (CIA) of a business and its users [65]. While numerous static (white-box) analysis and dynamic (black-box) analysis tools exist for identifying vulnerabilities in a system [66, 67], they have become far less effective due to the increasing complexity of web applications, their dependency on an array of downstream technologies, and the limited development and deployment time [68].

In [69], researchers propose the use of moving target defense (MTD) for web-application security. The MTD continually moves the technologies in the web-stack

to (1) reduce the time-window of attacks for an adversary, making the web-application more resilient while (2) ensuring expected services are always available to the legitimate users. However, *The design of good quality switching strategies is left as an open problem.* In this chapter, we show that a good movement strategy is key to effectively maximize the security of MTD-based web-applications.

To design effective switching policies, we first model the interaction between the defender $\mathcal{D}$ and the attacker $\mathcal{A}$ as a Bayesian Stackelberg Game (BSG) [61]. The Bayesian nature stems from the uncertainty over attacker types that may attack the web-application. The Stackelberg nature is inherent in the context of a web-application security setting because a defender $\mathcal{D}$ has to deploy an MTD-based web-application first (making them the leader) that will eventually, with time on the attacker's side, be fingerprinted by the adversary (making them the follower). Second, we use real-world attack data to determine various parameters of our game-theoretic model. We propose a formal framework to define attacker types and can automatically generate their pure-strategy sets by mining the National Vulnerability Dataset (NVD) for Common Vulnerabilities and Exploits (CVEs). The rewards of our game are obtained by leveraging crowd-sourced knowledge of security experts encoded in the Common Vulnerability Scoring System (CVSS). Third, we address a novel challenge offered by MTD systems– the performance impact incurred due to a move/shift action. While some move actions allow the system to gracefully handle the change in code flow for the incoming request, others might drop the packet and expect the user to retry thereby impacting the customer's experience. In this regard, we find that existing equilibrium computation methods, developed extensively for physical security problems [70], do not account for the cost of switching from one configuration to another. Thus, we adapt existing optimization methods that infer movement strate-

Figure 3.1: A moving target web-application system requires an effective switching strategy.

gies at Stackelberg Equilibrium to penalize costly switches and encourage moves that have less impact on the performance of the system alongside security considerations.

Beyond the design of effective switching policies, we discuss two novel challenges that arise in the use of MTD for web-application security. First, all known vulnerabilities of a web application cannot be fixed in an instant owing to functionality constraints, limited developer bandwidth, and ongoing discovery of novel vulnerabilities. Thus, prioritizing what needs to be patched first becomes crucial. We show that the increased complexity of MTD systems coupled with uncertainty about attacker types exacerbates the difficulty of prioritizing known vulnerabilities. In this regard, we define this problem formally and propose a preliminary solution. Second, the actual uncertainty over attacker types modeled by our framework may differ from reality. To understand the effect of this difference, we conduct experiments to measure the robustness of existing baselines and the inferred movement strategies.

We introduce the domain of MTD systems for web applications in Section 3.1. In Section 3.2, we model the MTD as a Bayesian game– we define the attacker types, propose rules to construct their pure-strategy sets and methods of obtaining rewards

by using established security metrics. To find an effective switching strategy, in Section 3.3 we propose a solver that maximizes system security while accounting for switching costs and empirically study the efficacy and robustness of the generated strategies in Section 3.4, comparing it to the state-of-the-art. We then formulate the problem of identifying critical vulnerabilities in Section 3.4. We briefly discuss related work in Section 3.5 before concluding the chapter in Section 3.6.

3.1 Moving Target Defense for Web Applications

In this section, we present a brief overview of the MTD system in the context of web applications. We instantiate the notations defined in Section 2.1 in regard to the particular setting at hand.

The Configuration Set C A configuration set for a web-application stack is denoted as $C = C_1 \times C_2 \cdots \times C_n$, a Cartesian product of the technologies in the n-technological stacks. Here, C_i denotes the set of technologies that can be used in the i-th layer of the application stack. A *valid* configuration $c \in C$ is thus, an n-tuple that preserves the system's operational goals.

Consider a web application that has two stacks (i.e. $n = 2$) where the first layer denotes the coding language used to make the web-application and the second layer denotes the database technology used to store or retrieve the data handled by this application. Say, the technologies used in each layer are $C_1 = \{\text{Python, PHP}\}$ and $C_2 = \{\text{MySQL, postgreSQL}\}$. Then, a valid configuration can be $c = (\text{PHP, MySQL})$. The *diversity* of an MTD system, which is the number of valid configurations, is upper-bounded by the cardinality of $C (= |C_1| \cdot |C_2| \ldots |C_n|)$.

Attacks Software security is defined in terms of three characteristics - Confidentiality, Integrity, and Availability [71]. In a broad sense, a cyber attack is defined as an *act* that compromises any of the aforementioned characteristics of a cyber system. The National Vulnerability Database (NVD) is a public directory consisting of Common Vulnerabilities and Exposures (CVEs). For each CVE, the database lists a set of technologies that are vulnerable; this helps us identify attacks that can be used against our web-application system. (we use the terms *vulnerability* and *attack* interchangeably going forward).

The Movement Strategy M As described in Section 2.1, M in our case represents the decision making process for the defender– to select the next valid system configuration c' given the present deployed configuration c (where both $c, c' \in C$). An added complexity in the domain is the presence of switching costs that are incurred when the system moves from a configuration c to another configuration c'. Thus, the aim of a good strategy is to maximize the security provided by an MTD system while trying to minimize the cost of switching. State-of-the-art MTD systems in web-applications use a uniformly random switching strategy (i.e. select any $c \in C$ with probability $1/|C|$); this assumes that switching between any two configurations incurs the same cost by default [72].

We now describe game-theoretic modeling that can be leveraged to generate switching strategies for the MTD system. With this model, we will be able to characterize optimal movement and, in turn, show that a uniformly distributed switching strategy is sub-optimal.

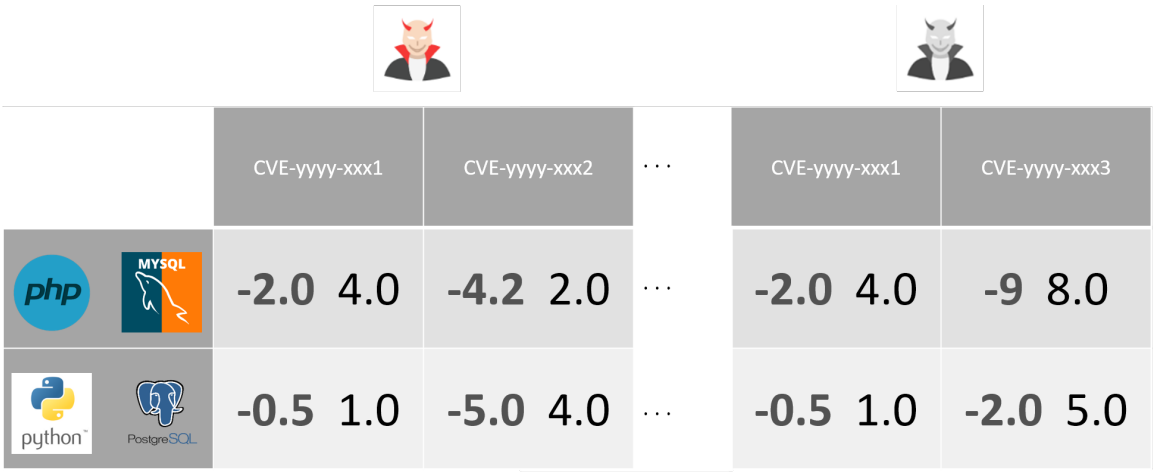


Figure 3.2: An example game between the defender with two valid configurations and an attacker who can be one of two types.

3.2 Game Theoretic Modeling

In this section, we model the Moving Target Defense as a normal-form Bayesian Game. An example of the formulated normal-form game is shown in Figure 3.2.

Agents and Agent types As discussed before, The two players in our game ($N = 2$) are the defender $\mathcal{D}$ and the attacker $\mathcal{A}$. The set θ_i is the set of types for player i ($= \{\mathcal{D}, \mathcal{A}\}$). Thus, $\theta_\mathcal{D}$ and $\theta_\mathcal{A}$ denotes the set of defender and attacker types respectively. The $j-$th attacker type is represented by $\theta_{\mathcal{A}j}$.

When an attacker attacks an application, its beliefs about what (resource/data) is most valuable to the application owner (defender) remains consistent. Thus, we assume that the attacker knows that there is only one type of defender when they attack a particular web application. Thus, we have $|\theta_\mathcal{D}| = 1$ and a sub-script based notation, introduced in the case of the attacker, in no longer necessary.

We consider a finite number of attacker types where each is defined in our model using the following three tuples,

$$\theta_{\mathcal{A}i} = \langle name, \{(expertise, technologies) \dots\}, probability \rangle$$

where the second field is a set of two-dimensional values that express an attacker's expertise ($\in [0, 10]$) in a technology. The rationale for using values in this range stems from the use of the Common Vulnerability Scoring System (CVSS) described later. Lastly, the set of attacker types have a discrete probability distribution associated with it. The probability $P_{\theta_{\mathcal{A}j}}$ represents the defender's belief about the attacker type $\theta_{\mathcal{A}j}$ attacking their application. Obviously, the probability values of all attacker types sum up to one $\sum_{\theta_{\mathcal{A}j} \in \theta_{\mathcal{A}}} P_{\theta_{\mathcal{A}j}} = 1$.

Note that one can define attacker expertise over a '*category of attacks*' (like *'BufferOverflowAttacks'*) instead of *technology* specific attacks. We feel the latter is more realistic for our domain. This definition captures the aspect that an attacker type can have expertise in a set of *technologies*. Since these attacker types and the probability distribution over them are application-specific, it is defined by a domain expert and taken as an input to our proposed model. For instance, a defender using a no-SQL database in all configurations of his MTD system assigns zero probability to an '*SQL_database*' attacker type because none of their attacks can compromise the security of his present system.

The assumption that the input probability distribution over all the attacker types can be accurately specified is a strong one. We will later discuss how errors in judgment can affect the effectiveness of a switching strategy and define a measure to capture the robustness of the generated policy in such circumstances.

Agent actions We define $A_{\mathcal{D}}$ as a finite set of actions available to player $\mathcal{D}$. The defender's action set $A_{\mathcal{D}}$ comprised of switch actions, each corresponding to a valid configuration, $c \in C$ of the web application. Hence, $A_{\mathcal{D}} = C$ and $|A_{\mathcal{D}}|$ is bounded by the number of valid configurations. As mentioned earlier, the number of valid configurations $|C|$ is often lower than $|C_1| \cdot |C_2| \ldots |C_n|$ in real-world settings because a technology used in layer i may not be compatible with a particular technology used in layer j ($\neq i$) rendering that configuration *invalid*.

For the attacker, $A_{\theta_{\mathcal{A}}}$ represents the set of all attacks used by at least one attacker type. A particular attack a belongs to the set $A_{\theta_{\mathcal{A}}}$ if it affects at least one of the technologies used in the layers for our web application ($C_1 \cup C_2 \cdots \cup C_n$).

We now define a function $f : (\theta_{\mathcal{A}t}, a) \rightarrow \{1, 0\}$ for our model. The function implies an attack a belongs to an attacker type $\theta_{\mathcal{A}t}$'s arsenal $A_{\theta_{\mathcal{A}t}} (\subseteq A_{\theta_{\mathcal{A}}})$ if the value of the function is 1. This function value is based on (i) the expertise of the attacker type contrasted with the 'exploitability' necessary to execute the attack, and (ii) the attacker's expertise in the technology for which the attack can be used. We provide a concrete definition for the function f after elaborating on what we mean by the exploitability of an attack.

For (almost all) CVEs listed in the NVD database, we have a six-dimensional CVSS v2 vector representing two independent scores – Impact Score (IS) and Exploitability Score (ES). For an attack action a, let ES_a ($\in [0, 10]$) represent the ease of exploitability (higher is tougher). Each attack also has a set of technologies it affects, say T^a.

Let us say that the set of technologies an attacker type $\theta_{\mathcal{A}t}$ has expertise in, is T_t. We can now define the function f as,

$$f(\theta_{\mathcal{A}t}, a) \quad = \quad \begin{cases} 1, & \textit{iff } T_t \cap T^a \neq \phi \ \wedge ES_a \leq \textit{expertise}_t \\ 0 & \textit{otherwise} \end{cases}$$

Where the condition $ES_a \leq \textit{expertise}_{\theta_{\mathcal{A}t}}$ must hold for all the technologies $\in T_t \cap T^a$.

Rewards Now that we have attack sets for each attacker type, the general reward structure for the proposed game is defined as follows:

$$R^{\mathcal{A}}_{a,\theta_{\mathcal{A}i},c} = \begin{cases} +x_a & \text{if } a \subset \upsilon(c) \\ -y_a & \text{if } a \text{ can be detected or } a \subset c' \\ 0 & \text{otherwise} \end{cases}$$

$$R^{\mathcal{D}}_{a,\theta_{\mathcal{A}i},c} = \begin{cases} -x_d & \text{if } a \subset \upsilon(c) \\ +y_d & \text{if } a \text{ can be detected or } a \subset c' \\ 0 & \text{otherwise} \end{cases}$$

where $R^{\mathcal{A}}_{a,\theta_{\mathcal{A}i},c}$ and $R^{\mathcal{D}}_{a,\theta_{\mathcal{A}i},c}$ are the rewards for the attacker $\mathcal{A}$ and the defender $\mathcal{D}$ respectively when the attacker type $\theta_{\mathcal{A}i}$ uses an attack action a against a configuration c ($\in C$). The function $\upsilon(c)$ represents the set of security vulnerabilities (CVEs) that configuration c has. Also, c' refers to a honey-net configuration. A honey-net is a configuration setup with intentional vulnerabilities for trapping attackers.

The reward values for $\mathcal{D}$ when an attacker does not attack (chooses the NO-OP action), is zero. Moreover, a defender gets zero rewards for successfully defending a system. We reward them positively only if they are able to reveal some information about or catch the attacker without impacting operation requirements for the non-malicious users. They get a negative reward if an attacker successfully exploits their system.

To instantiate the variables x_a, y_a, x_d and y_d, we make use of CVSS(v2) metric. This metric provides the Impact (IS) and Exploitability Scores (ES), stated above, which are combined to calculate a third score called Base Score (BS) [73]. Using these, we now define the following:

$$\begin{aligned} x_d &= -1 * IS \\ x_a &= BS \end{aligned}$$

Note that the BS considers both the impact and the exploitability scores. When the IS for two attacks is the same, the one that is easier to exploit gets the attacker a higher reward value. The ease of an attack can be interpreted in terms of the resource and effort spent by an attacker for an attack *vs.* the reward (s)he attains by harming the defender. Although the robustness of our framework provides provisions for having y_d and y_a, detecting attacks on a deployed system or setting up honey-nets is non-trivial in present web application systems. Hence, there are no actions where values of y_d or y_a are required in our present application.

Before moving on to the next section, we describe briefly the security aspects that the two independent scores– IS and ES– seek to capture in the context of real-world software systems. For this purpose, we first define the six independent feature values that are necessary to generate these scores.

- ***Access Vector (AV)*** is dependent on the amount of access an attacker needs to exploit a vulnerability. An attack that needs physical access to a system will have a lower score than one that can be exploited over the Internet.

- ***Access Complexity (AC)*** represents the complexity of exploiting an attack. A buffer overflow attack on an Internet service is less complex than an e-mail client vulnerability in which a user has performed attachment downloads followed by executing it and hence has lower AC value.

- ***Authentication (Au)*** level required to execute the attack. If no sign-up account is required to exploit the system, this value is high. In contrast, if one needs multiple accounts to exploit the vulnerability, the value is low.

- ***Confidentiality Impact (C)*** scores are low if only some (non-relevant) information gets leaked. The highest impact occurs when say, the entire database is compromised if the vulnerability is successfully exploited.

- ***Integrity Impact (I)*** refers to the attacker's power to modify files or behavior of a system if he executes the exploit successfully. A higher value indicates more power.

- ***Availability Impact (A)*** represents the power of a successful exploit to bring down the availability of a system. A successful Denial of Service (DoS) that brings down an application server, will have a high impact.

From these values, one can obtain the two independent scores using the following formulas,

$$\begin{aligned} ES &= 20 * (AV) * (AC) * (Au) \\ IS &= -10.41 * (1 - (1 - C)(1 - I)(1 - A)) \end{aligned}$$

The CVSS values are generated by security experts across the globe. A rigorous treatment of how one should determine these values can be found in [73].

Our model takes a time range as input. It then parses all the CVEs (a) from the NVD in that time range to finally filter out the ones that can affect at least one of the configurations in our system ($a \subset \upsilon(c_i)$). Note that CVEs older than a particular time become irrelevant when composing an attacker type's pure strategy set against a modern-day MTD system because (1) they either have no effect on the updated

versions of the technologies or (2) have popular solutions available to developers at the time of application development. For our application, we obtain this input range from our security experts. The normal-form representation of an example game between two attacker types is shown in Figure 3.2.

Switching Cost The switching costs can be represented by a $K^{n \times n}$ matrix where the n rows (and columns) denote the n system configurations. The cell K_{ij} denotes the cost of switching when the defender moves from configuration i to configuration j. As mentioned earlier, the values in K are all non-negative. Our security experts, who have written the code to automatically move from one configuration to another, hand-coded these values in each cell of the matrix. We provide some guidance in choosing these values here and give a concrete example of how we selected these for our application later.

If there is no common technology between configurations c and c' involved in a switch operation, the cost will be large. Also, switching technologies in a specific layer may incur more cost than switching technologies in other layers. In the developed MTD system, we find that switching between databases incur large costs because the structure of the data needs to be changed for shifting, and the time required to copy huge amounts of data from one database to another must also be accounted for.

The matrix K for our system turns out to be symmetric, i.e. $K_{ij} = K_{ji} \; \forall \; i, j \in \{1, \ldots n\}$. Also, $K_{ii} = 0$, which implies that there is no cost if no configuration switch occurs. Note that although our security experts think this is the structure of rewards for the developed system, the modeling is generic enough to allow for asymmetric costs. Lastly, we choose the values of K_{ij} in the range $[0, 10]$. The reason for this upper bound becomes clear in the upcoming section.

3.3 Switching Strategy Generation

In this section, we first introduce the notion of Stackelberg Equilibrium for our security game. This is the defender's movement strategy that maximizes their reward (and thus, (only) the security of the system). We then briefly introduce existing optimization methods, relevant to our particular set, that can use to generate the equilibrium strategy. Given it cannot model the switching costs, it produces movement strategies that heavily impact performance. Finally, we describe our optimization method that addresses this concern.

3.3.1 Stackelberg Equilibrium

The strategy generated for the designed game needs to capture the reconnaissance aspect. Note that the game starts only after the defender has deployed the web application, acting as a leader. Given that an attacker can observe the switch moves (by probing) and in due time learn the switching strategy (as $|C| \ll \infty$) of the defender (using Maximum Likelihood Expectation). [1] Thus, the defender needs to select a strategy that maximizes their reward in this game, subject to the threat-model where the attacker knows their (mixed) strategy. [2] This is exactly the problem of finding the Stackelberg Equilibrium in a Bayesian Game [74]. The resulting mixed strategy is the switching strategy for the defender. Unfortunately, as stated in Chapter 2, this problem is *NP-hard* owing to the uncertainly over multiple attacker types [75].

Before we find a *Strong* Stackelberg Equilibrium (SSE) for our proposed game, we state a couple of well-founded assumptions made. First, an attacker chooses a

[1] Consider a $|C|$-sided dice. The problem is similar to finding the probability of each side coming up given a set of rolls.

[2] This implies that a pure-strategy, unless it is completely secure, makes the attacker jobs easier. Also, modern web-applications employ a pure-strategy.

pure strategy, i.e., a single attack action that maximizes their reward. As discussed in Chapter 2, this assumption is not-limiting because for every mixed strategy of the defender at SSE, the attacker has a pure strategy [76]. Second, the notion of Strong Stackelberg Equilibrium states that if there are two (or more) pure-strategies that yield the same reward to $\mathcal{A}$ but yield different rewards to $\mathcal{D}$, $\mathcal{A}$ chooses the one that favors $\mathcal{D}$. While this notion does not sit well in an adversarial setting, it is popular because (1) an SSE is guaranteed to exist in a normal-form BSG and (2) it is always possible for $\mathcal{D}$ to play a slightly perturbed equilibrium strategy that forces the attacker to play in their favor.

To solve for the optimal mixed strategy, one can use the Decomposed Optimal Bayesian Stackelberg Solver (DOBSS) [61]. This optimizes the expected reward of the defender over all possible mixed strategies for the defender ($\vec{x}$), and pure strategies for each attacker type ($\vec{q}^{\,\theta_{\mathcal{A}i}}$) given the attacker type uncertainty ($\vec{P}_{\theta_{\mathcal{A}i}}$). We now define the objective function of the Mixed Integer Quadratic Program (MIQP).

$$\max_{x,q,v} \sum_{c \in C} \sum_{\theta_{\mathcal{A}i} \in \theta_{\mathcal{A}}} \sum_{a \in A_{\theta_{\mathcal{A}i}}} P_{\theta_{\mathcal{A}i}} R^{\mathcal{D}}_{a,\theta_{\mathcal{A}i},c} \; x_c q_a^{\theta_{\mathcal{A}i}} \tag{3.1}$$

We observe that solving the MIQP version is more efficient (in computation time and memory usage) than solving the Mixed Integer Linear Program (MILP) version of DOBSS. We hypothesize that this seemingly surprising phenomenon is caused because the MILP formulation results in an increase in the dimensions of the solution space. Precisely, the MIQP solves for $|A^{\mathcal{D}}| + \sum_{\theta_{\mathcal{A}i} \in \theta_{\mathcal{A}}} \sum_{a_j \in A_{\theta_{\mathcal{A}i}}} |a_j|$ variables where as the MILP solves for $|A^{\mathcal{D}}| * \sum_{\theta_{\mathcal{A}i} \in \theta_{\mathcal{A}}} \sum_{a_j \in A_{\theta_{\mathcal{A}i}}} |a_j|$ variables. While we do not have a theoretical proof to support this claim, we conduct a set of empirical evaluations to test our hypothesis.

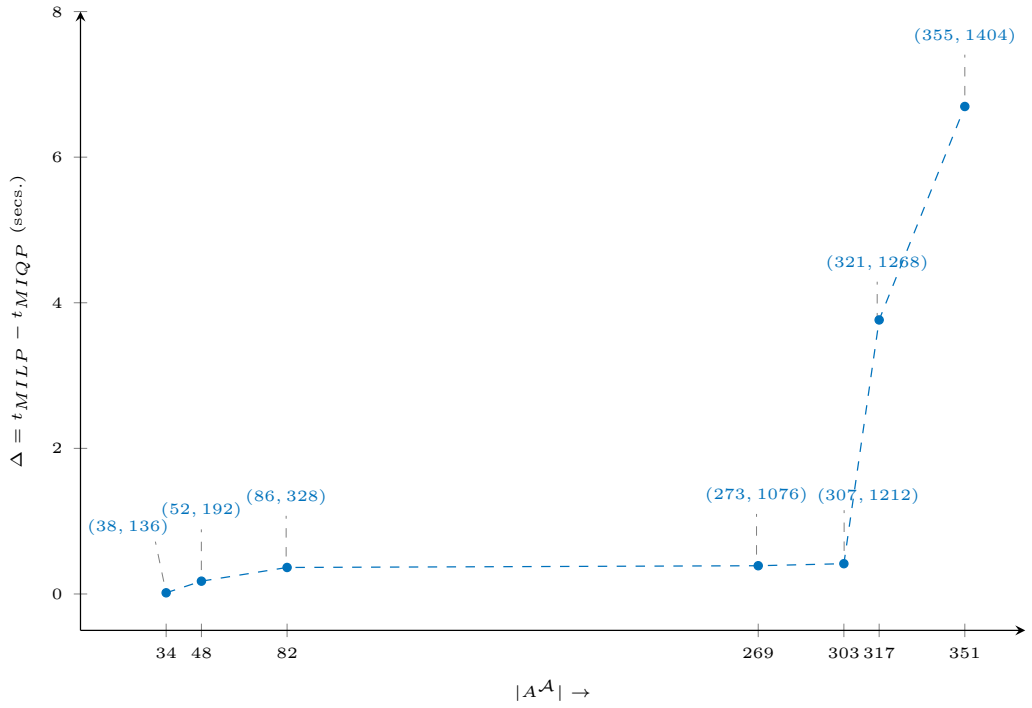


Figure 3.3: The MILP takes more time than the MIQP formulation as the number of attack actions (for all types combined) increases. The numbers in bracket indicate $|A^{\mathcal{A}}|$ in the (MIQP, MILP) for each instance.

In Figure 3.3, preliminary results on our MTD domain show that our hypothesis holds. We see that as the number of attack variables (plotted on the x-axis) increases, the time difference between $\Delta = t_{MILP} - t_{MIQP}$ increases. The points on the x-axis are obtained by considering all possible non-empty subsets of attacker types described later in Table 3.2 (and all types being equally probable). We consider an average of three runs to calculate the Δ values (plotted on the y-axis). Even for the smallest instance with 34 attack actions, the time difference is positive, indicating that the MILP formulation for DOBSS takes more time than the MIQP formulation. While there is a constant increase in the Δ as $|A^{\mathcal{A}}|$ increase, this is hardly noticeable for $|A^{\mathcal{A}}| \leq 303$ because millisecond time difference seems negligible in comparison the

time difference (in multiple seconds) seen as $|A^{\mathcal{A}}| > 303$. The tuples over the points indicate the number of optimization variables in the MIQP followed by the number of variables in the MILP. We notice a sudden jump in Δ as the number of total attack actions (across attacker types) increases from 303 to 317.

Owing to this phenomenon, we will refer to the MIQP (as opposed to the MILP) formulation, both presented in [61], whenever referring to the DOBSS solver going forward. Lastly, we note that Equation 3.1 does not consider that switching costs between defender strategies; it inherently assumes that the costs for movement, regardless of the switching configurations, are uniform.

3.3.2 Incorporating Switching Costs

As defined in the last section, the cost for switching from a configuration i to a configuration j can be represented as K_{ij}. The probability the system is in configuration i and then switches to configuration j is $x_i \cdot x_j$. Thus, the cost incurred by the defender for a switch action from i to j is $K_{ij} \cdot x_i \cdot x_j$. The expected cost for any switch action is $\sum_{i \in C} \sum_{j \in C} K_{ij} \cdot x_i \cdot x_j$. To account for cost, we can subtract this expression from the objective function in Equation 3.1 with a cost-accountability factor α (≥ 0) to obtain

$$\max_{x,q,v} \sum_{c \in C} \sum_{\theta_{\mathcal{A}i} \in \theta_{\mathcal{A}}} \sum_{a \in A_{\theta_{\mathcal{A}i}}} P_{\theta_{\mathcal{A}i}} R^{\mathcal{D}}_{a,\theta_{\mathcal{A}i},c} \, x_c q_a^{\theta_{\mathcal{A}i}} - \alpha \cdot \sum_{i \in C} \sum_{j \in C} K_{ij} \cdot x_i \cdot x_j$$

Unfortunately, this results in a Bilinear Mixed Integer Programming problem, which is not convex. To ameliorate this problem, we introduce new variables w_{ij} that represent an approximate value of $x_i \cdot x_j$. We first use the piece-wise-linear McCormick envelopes to design a convex function using these w_{ij}-s that are known to generate good estimates and thus, a satisfying and yet, scalable solution to this problem [77]. Along with these constrains, we now describe the final MIQP convex optimization

problem as follows,

$$\max_{x,q,v} \sum_{c\in C} \sum_{\theta_{\mathcal{A}i}\in\theta_{\mathcal{A}}} \sum_{a\in A_{\theta_{\mathcal{A}i}}} P_{\theta_{\mathcal{A}i}} R^{\mathcal{D}}_{a,\theta_{\mathcal{A}i},c}\; x_c q_a^{\theta_{\mathcal{A}i}} \;-\; \alpha \cdot \sum_{i\in C}\sum_{j\in C} K_{ij} w_{ij} \tag{3.2}$$

$$\begin{aligned}
s.t. \quad \sum_{c\in C} x_c &= 1 && (3.3)\\
\sum_{a\in A_{\theta_{\mathcal{A}i}}} q_a^{\theta_{\mathcal{A}i}} &= 1 && (3.4)\\
0 \le v^{\theta_{\mathcal{A}i}} - \sum_{c\in C} R^{\mathcal{A}}_{a,\theta_{\mathcal{A}i},c} x_c &\le (1 - q_a^{\theta_{\mathcal{A}i}})M && (3.5)\\
w_{ij} &\ge 0 \;\forall\; i,j && (3.6)\\
w_{ij} &\le x_i \;\forall\; i,j && (3.7)\\
w_{ij} &\le x_j \;\forall\; i,j && (3.8)\\
\sum_{j\in C}\sum_{i\in C} w_{ij} &= 1 \;\forall\; i,j && (3.9)\\
\sum_{j\in C} w_{ij} &= x_i \;\forall\; i && (3.10)\\
\sum_{i\in C} w_{ij} &= x_j \;\forall\; j && (3.11)
\end{aligned}$$

$$x_c \in [0\ldots 1],\; q_a^{\theta_{\mathcal{A}i}} \in \{0,1\},\; v^{\theta_{\mathcal{A}i}} \in \mathcal{R}$$

$$\forall\; c \in C,\; \theta_{\mathcal{A}i} \in \theta_{\mathcal{A}},\; a \in A_{\theta_{\mathcal{A}i}}$$

where M is a large positive number. $\vec{q}^{\theta_{\mathcal{A}i}}$ and $v^{\theta_{\mathcal{A}i}}$ give the pure strategy and its corresponding reward for the attacker type $\theta_{\mathcal{A}i}$, and $\vec{x}$ gives the mixed switching strategy for the defender. Constraint 3.5 solves the dual problem of maximizing rewards for each attacker type ($v^{\theta_{\mathcal{A}i}}$) given the defender's strategy which ensures that attackers select the best attack action. The constrains 3.6, 3.7, and 3.8 represent the McCormick envelope that provides lower and upper bounds on each w_{ij}.

We now introduce more constrains on the values w_{ij} relevant to our problem to generate tighter approximations for the value $x_i \cdot x_j$. Since we consider all possible

switches, $\sum_{j \in C} \sum_{i \in C} x_i \cdot x_j = 1$. This is enforced by 3.9. Lastly, for each i, $\sum_{j \in C} x_i \cdot x_j = x_i \cdot (\sum_{j \in C} x_j) = x_i$. This is represented by the constraints 3.10 and 3.11.

The optimization problem defined in Equation 4.5 guarantees a good strategy for the defender when the MTD application is being deployed. After that, in a repeated game consideration, this strategy becomes sub-optimal because the w_{ij}–s does not take into account the defender's decision in the previous round (given that some element of $\vec{x}$ is 1 and all the rest 0). To address this, the expression $x_i \cdot x_j$ would have to be $x_i^{(t)} \cdot x_j^{(t+1)}$ where $x_i^{(t)} = 1$ for the i-th configuration that was deployed at time t and 0 for the others. The variables here are only $x_j^{(t+1)}$ $\forall j$, which can easily be found by solving the following optimization problem,

$$\max_{x,q,v} \sum_{c \in C} \sum_{\theta_{\mathcal{A}i} \in \theta_{\mathcal{A}}} \sum_{a \in A_{\theta_{\mathcal{A}i}}} P_{\theta_{\mathcal{A}i}} R^{\mathcal{D}}_{a,\theta_{\mathcal{A}i},c} \, x_c q_a^{\theta_{\mathcal{A}i}} - \alpha \cdot \sum_{i \in C} \sum_{j \in C} K_{ij} \cdot x_i^{(t)} \cdot x_j$$

with the domain constrains and constraints (3), (4), and (5). Note that this is a convex function because $x_i^{(t)}$ are constants. Thus, the defender can now obtain the best strategy after each round. Although a Markov Game formulation of this setting with states corresponding to the number of configurations in C is possible, we don't consider it because dealing with this the uncertainty over attacker types adds an extra layer of complexity.

If we now allow the maximum cost of switching to be 10, we can see that the values for the switching cost become comparable in magnitude to the defender's utility values. This helps us to provide semantic meaning for the cost-accountability factor, α. The first term in the objective function seeks to maximize the defender's reward, which in turn maximizes the security of the web application. The second term, on the other hand, seeks to reduce the expected cost of the switching actions. Thus, if a defender selects a low α value, they give more significance to the first term, i.e.

	PHP, MySQL	Python, MySQL	PHP, postgreSQL	Python, postgreSQL
PHP,MySQL	0	2	6	10
Python,MySQL	2	0	9	5
PHP,postgreSQL	6	9	0	2
Python,postgreSQL	10	5	2	0

Table 3.1: Switching costs for our system.

security. To provide a sense to the reader, we later show in the experimental section, how strategies and reward values are effected with changing alpha values.

3.4 Empirical Evaluation

The goal of this section is to answer three key questions. First, does our proposed Bayesian Stackelberg Game (BSG) model generate better strategies than the state-of-the-art? Second, can we effectively compute the set of critical vulnerabilities? Third, who are the sensitive attacker types, and how robust is our proposed strategy?

Test Bed Description

To answer the questions mentioned above, we develop a real-world MTD web application (Figure 3.1) with 2 layers. The key idea of applying MTD to web applications requires you to have several versions of the same system, each written in either a different language, using a different database, etc. This diversity is not ubiquitous in legacy web applications, due to the cost, time, and resources required to build several versions of the same web application. To aid this, we developed a framework to

Name	(Technologies, Expertise)	Prob.	$\lvert A_{\theta_{\mathcal{A}i}} \rvert$
ScriptKiddie (SK)	(PHP,4), (MySQL,4)	0.15	34
Database Hacker (DH)	(MySQL,10), (postgreSQL,8)	0.35	269
Mainstream Hacker (MH)	(Python,4), (PHP,6), (MySQL,5)	0.5	48

Table 3.2: The attacker types with the number of attack actions.

automatically generate the diversity necessary for this web application. The current prototype is able to convert a web application coded in Python to an equivalent one coded in PHP, and vice versa, as well as a web application using a MySQL database to an identical version that uses PostgreSQL, and vice versa. In the future, as more and more variations are developed, the set of defender's actions will increase.

The present set of valid configurations for our system is $C = \{$(PHP, MySQL), (Python, MySQL), (PHP, postgreSQL), (Python, postgreSQL)$\}$. The costs for switching between configurations is shown in Table 3.1. These cost values generated are based on the following considerations:

• Switching between different languages while keeping the same database dialect incurs minimal cost. Workload is primarily rerouting to the correct server with the source code.

• Switching between different database dialects while keeping the same language incurs a higher cost due to the conversion required for the database structure and its contents. One also has to account for copying large amounts of data to the database used in the current system configuration.

• Switching between different database dialects AND different languages incur the highest cost due to the combination of the costs of the database switch as well as the penalty for rerouting to the correct server with the source code.

The attacker types along with the attack action set size are defined in Table 3.2. We mined the NVD for obtaining CVE data from January 2013 to August 2016 to generate these attack sets. When the stakes of getting caught are too high for an attacker type, (s)he may choose not to attack. Hence, we have a *NO-OP* action for each attacker type.

The optimization problems for the experiments were solved using Gurobi on an Intel Xeon $E5$ $2643v3$@3.40GHz machine with 6 cores and 64GB of RAM.

3.4.1 Strategy Evaluation

We evaluate our method using Bayesian Stackelberg Games on our real-life web application against the Uniform Random Strategy (URS), which is the state-of-the-art in such systems [72]. We plot the values of the objective function in Equation 4.5 for both the strategies as α varies from 0 to 1. For URS, we use the exact values of $w_{ij} = 0.25 * 0.25 = 0.0625 \ \forall \ i, j$. The plot is shown in Figure 3.4. Both are straight lines because although the value of α changes, the strategy for URS is the same (by definition) and the one generated by BSG also remains the same. The latter case came as a surprise to us initially. On further investigation, we noticed that in the formulated game for our web-application, the Stackelberg Equilibrium for our application (luckily) coincides with the least switching cost strategy.

These attacker and defender strategies are shown in Table 3.3 along with the value of the defender's reward (i.e. the first term in the objective function in Equation 4.5). Notice that, not only is the mixed strategy generated by BSG more secure than URS,

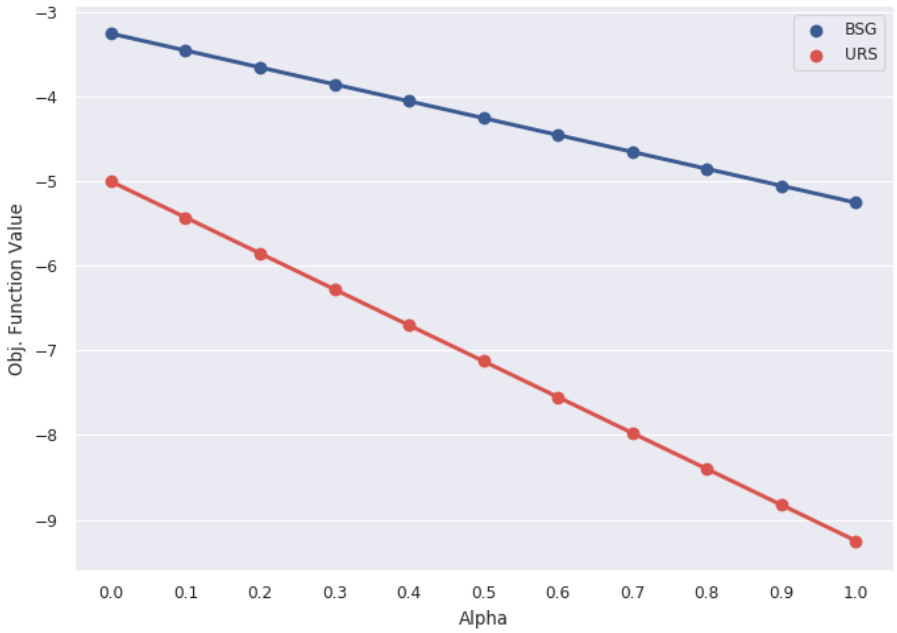


Figure 3.4: Objective function values for Uniform Random Strategy *vs.* Bayesian Stackelberg Game with switching costs as α varies from 0 to 1.

it leverages fewer configurations than all valid configurations $|C| = 4$ the system has to offer. Our result is consistent with earlier work in cybersecurity which shows that a higher number of configurations does not imply better security guarantees [78].

Studying the effect of α-values

To empirically show that our solver is actually considering costs of switching, we change the value for switching from (PHP, PostgreSQL) to (Python, postgreSQL) and vice-versa from 2 (yellow boxes in Table 3.1) to 10. We plot this scenario in Figure 3.5. As soon as $\alpha \geq 0.4$, the BSG generates $(0.25, 0.25, 0.25, 0.25)$ (which is URS) as the most optimal strategy. After analysis, we note that this happens because the most powerful attack actions in the arsenal of the attacker types are for the systems (PHP, MySQL) and (Python, MySQL). When, one does not prioritize

Method	Mixed Strategy	Defender's Reward	Attack sets (SK, DH, MH)	
URS	(.25, .25, .25, .25)	-5	CVE-2016-3477, CVE-2016-3477	CVE-2015-3144,
BSG	(.0, .0, .5, .5)	-3.25	CVE-2014-0185, CVE-2014-0185	CVE-2014-0067,

Table 3.3: Comparison between the strategies generated by Uniform Random Selection (URS) *vs.* Bayesian Stackelberg Game (BSG).

switching costs ($\alpha \in \{0, 0.1, 0.2, 0.3\}$), the system keeps switching between the more secure configurations nullifying the good attacks of the attackers. As switching costs start to get more significant ($\alpha \in \{0.4, 0.5, \ldots 1.2\}$), the objective function value reduces if it sticks to the stronger configurations since switching costs are now high for these. It switches to the URS in this case. Beyond that, it switches to the strategy $(0.25, 0.5, 0, 0.25)$ as α keeps on increasing. When α becomes close to 2, it completely ignores the security of the system and tries to minimize the switching cost by proposing the strategy $(0.5, 0.5, 0, 0)$ as the cost for switching between (PHP, MySQL) and (Python, MySQL) is the least ($= 2$).

In Figure 3.5, we show the change in the values of the objective function with respect to α. At first, the BSG generates a better strategy when compared to URS. When the BSG strategy becomes the same as the URS (for $0.4 \leq \alpha \leq 1.2$), the objective function value for BSG becomes lower than URS. This is not surprising since BSG is merely trying to estimate the value $x_i \cdot x_j$ with the variables w_{ij}, whereas URS is using the exact value. As we increase α further, we are essentially discouraging switching in an MTD system since now the cost of switching becomes too high (URS is not affected by this).

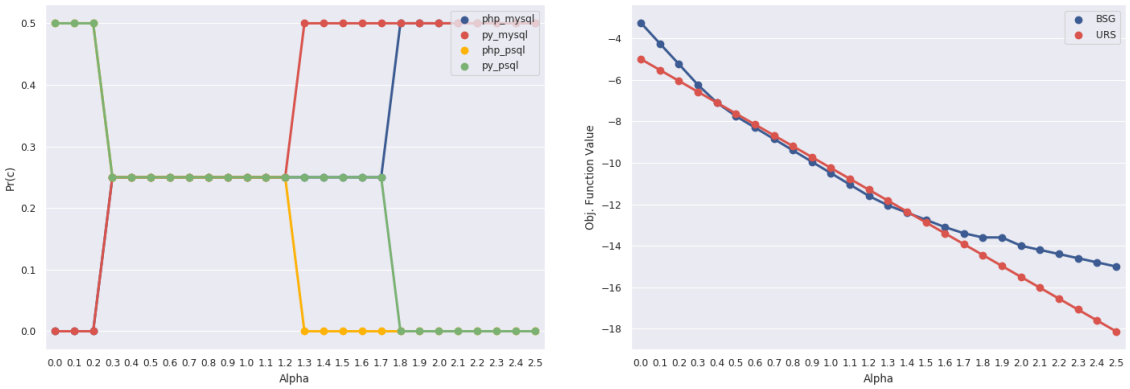


Figure 3.5: *Left:* Showcases the change in probabilities associated with a particular configuration. *Right:* Objective function values for Uniform Random Strategy *vs.* Bayesian Stackelberg Game with switching costs as α varies from 0 to 2.5 when the cost of switching shown in Table 3.1 are same in all cases except the values in the yellow boxes (which is made to be 10).

3.4.2 Identifying Critical Vulnerabilities

In real-world development teams, it is impossible to patch all the vulnerabilities, especially in a system with so many technologies. In current software systems, given a set of vulnerabilities, a challenging question often asked is which vulnerabilities should one fix to improve the security? For an MTD system, this becomes a tough problem since the defender needs to reason about multiple attacker types – their probabilities and attack actions. For a given k, the set of k fixed vulnerabilities which result in the highest gain in defender reward is termed as the k critical vulnerability set.

To address this problem, we remove each k-sized attack set from the set of all attacks ($A^{\mathcal{A}}_{-k} = A^{\mathcal{A}} \setminus D \; \forall \; D \subset A^{\mathcal{A}} \; \& \; |D| = k$) and evaluate the objective function (Equation 4.5). The set(s) $A^{\mathcal{A}}_{-k}$ that yield the highest objective values, provide the vulnerabilities D that should be fixed.

k	Critical Vulnerability Sets	Objective Value	CPU Time
1	{(CVE-2014-0185)}	-2.435	3m 15s
2	{(CVE-2014-0185, CVE-2015-5652)}	-1.973	421m 27s

Table 3.4: Most critical vulnerability in the MTD system and the time required to generate it.

We studied this complicated behavior for some toy examples. An interesting phenomenon we noticed was that a k-set critical vulnerabilities ($k-$CV) is not always a subset of the $(k+1)-$CV [21]. Suppose we want to find three vulnerabilities that we want to fix. As it is not a super-set of the 2-CV, we need to solve this problem from scratch with $k = 3$. Note that there is going to be a combinatorial explosion here. As the value of k increases, we end up solving $|A^{\mathcal{A}}_{-k}| = \binom{|A^{\mathcal{A}}|}{k}$ MIQP problems to identify the $k-$CVs.

Critical Vulnerabilities in the Developed System

We start with $k = 1$ and increase the number of critical vulnerabilities by one in each step. The result remains the same for $\alpha \in [0, 1]$. Unfortunately, the brute force approach and the scalability of algorithms for solving normal extensive form BSGs proves to be a key limitation. This is unsurprising since the total number of unique CVEs spread among the attackers is 287. When $k = 3$, we end up solving $\binom{287}{3}$ optimization problems, failing to scale in both time and memory requirements. Thus, we only show critical vulnerabilities identified up to $k = 2$ (in Table 3.4) using $\alpha = 0.2$.

At present, we are trying to develop a single MIQP formulation that tries to approximately generate the k-CV set. To reduce the combinatorial explosion, we plan to use switch variables that can turn attack actions on and off. This comes at the cost of increasing the number of variables in the formulated optimization problem.

3.4.3 Robustness & Attacker Type Sensitivity

It is often the case that a web application administrator (defender) cannot accurately specify the probability for a particular attacker type. In this section, we see how this uncertainty affects the optimal rewards generated by the system. We provide a notion for determining sensitive attacker types and measuring the robustness of a switching strategy.

For each attacker type i, we vary the probability $P_{\theta_{\mathcal{A}i}}$ by $\pm x\%$ ($P^{new}_{\theta_{\mathcal{A}i}} = P_{\theta_{\mathcal{A}i}}(1 \pm \frac{x}{100})$) where x is the *sensitivity factor*, which can be varied from a low value to a high value as needed. Note that now $p = P_{\theta_{\mathcal{A}i}} \times \frac{x}{100}$ needs to be distributed amongst the probabilities of the remaining attacker types. To make sure that this distribution is done such that the sensitivity of attacker i actually stands out, we propose to distribute p amongst the other attacker types using a weighted model as per their existing probabilities as shown below. For attacker j ($\neq i$), its new probability would be:

$$P^{new}_{\theta_{\mathcal{A}j}} = P_{\theta_{\mathcal{A}j}}(1 \mp \frac{p}{\sum_{k(\neq i)} P_{\theta_{\mathcal{A}k}}}) \tag{3.12}$$

When $x\%$ is subtracted from the probability $\vec{P}_{\theta_{\mathcal{A}i}}$, then the sign in the above equation becomes positive, and vice-versa.

We now formally define the loss in reward to the defender as the probability distribution over the attacker types change. Let R_o be the overall reward for the defender when he uses the mixed strategy for the assumed (and possibly incorrect) model of attacker type uncertainty ($\vec{P}_{\theta_{\mathcal{A}}}$) on the true model ($\vec{P}^{new}_{\theta_{\mathcal{A}}}$). Let R_n be the

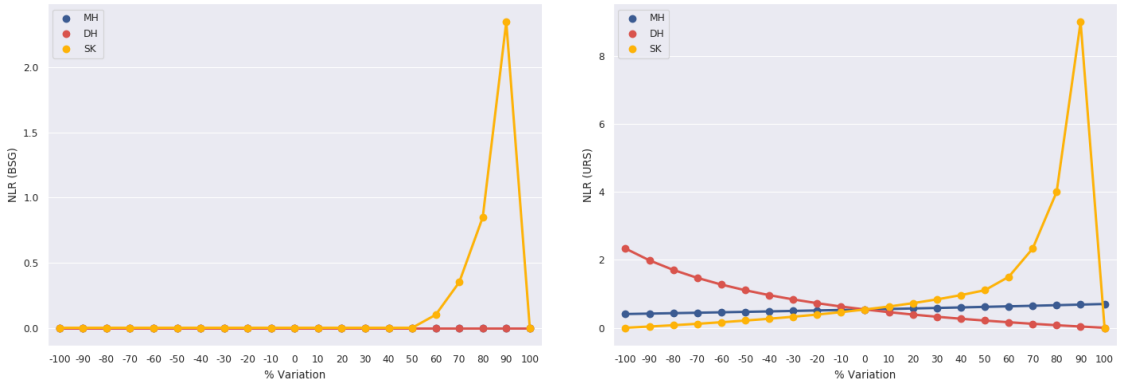


Figure 3.6: NLR values for BSG (left) and URS (right) when the attacker type's probabilities vary from 0% to 200% of its original value.

defender's optimal reward value for the true model. We compute the *Normalized Loss in Rewards (NLR)* for the defender's strategy as follows:

$$NLR = \frac{R_n - R_o}{R_n} \tag{3.13}$$

Note that NLR values are ≥ 0. Higher values of NLR represent more sensitive attacker types. Inaccurate probability estimates for the sensitive attackers can be detrimental to the security of our application. Note that lower NLR values indicate that a generated strategy is more robust.

Evaluation Based on the Developed System

We compute the attacker sensitivity for our system varying the probability of each attacker type from -100% to $+100\%$ (of its modeled probability) with 10% step sizes. We plot the results in Figure 3.6 using Equation 3.13. The Mainstream and Database hacker (MH & DH) is the least sensitive attacker types. The NLR values for both these attackers are 0. This is the case since the real-world attack action used by these types remains the same even when their probabilities change. On the other hand,

if the probability associated with the Script Kiddie (SK) is underestimated in our model, we see that the strategies deviate substantially from the optimal.

In this section, we use $\alpha = 0.2$. The max NLR for our BSG strategy is 2.35 *Vs.* 9 for URS. The average of the 60 NLR values is 0.061 for BSG and 0.88 for URS. These values indicate our model is more robust to variance in attacker type uncertainty than the present state-of-the-art.

3.5 Related Work

The majority of existing work in regards to the design of switching strategies for MTD systems is highly domain-specific. Thus, a trivial adaptation of these methods for MTDs in web application security is difficult. In this section, we discuss some of these works, highlighting their limitations in the domain of web applications and, thus, motivating the need for our solution.

The use of uniformly randomized switching strategies is considered to be the holy grail in theoretical formulations MTD systems [1]. We will demonstrate that our strategy, which can utilize domain-specific information, clearly outperforms such state-of-the-art. In the context of formal modeling, moving the underlying operating system has been previously viewed as Stochastic Game and trivial modifications to uniform random strategies such as selecting randomly between configurations different from the currently deployed configuration have been proposed [79, 22]. These works are highly curated for domains where an in-depth analysis of the code pertaining to individual configurations of the MTD system is possible and cease to be pragmatic for complex web applications. In [80], the MTD system is modeled as a game called PLADD; it is based on Flip-It games [44]. The game modeling assumes that different agents control a particular resource (eg. server) in different rounds of

the game. While the notion of continuous-time brings in an interesting and much-needed challenge to game-theoretic methods, this surrender of complete control of a resource to an attacker is impractical in most cyber-security applications (especially, web applications).

The closest work is perhaps the game-theoretic formulation in [78], where the authors consider a leader-follower setting for a dynamic platform defense. The goal is to come up with movement strategies that maximize a diversity metric based on *code similarity* among the different configurations of the MTD system. Such a diversity measure is (1) difficult to obtain in the domain of web applications and (2) can only be leveraged if the movement is considered for chosen layers of the web-stack. Further, the work neither considers the uncertainty in the attacker model nor explicitly models the costs for switching.

These aspects of uncertainty in the attacker model and attacker reconnaissance have been extensively handled in the context of Bayesian Stackelberg Games (BSG) [61], making them an appropriate choice for modeling MTD for web-applications. While existing solution methods in the physical security domains [70] provide scalable and equilibrium movement strategies for physical security settings, these do not consider the movement costs a defender may incur. Thus, we propose an optimization method that maximizes the defender's reward and minimizes the overall cost of switching between the various configurations of the web-application. The existing DOBSS solver [61] provides an essential stepping stone for the design of our approach.

The use of the Common Vulnerability Scoring System (CVSS) for rating attacks in cyber-security settings has been previously investigated in [81]. CVSS; we discuss in much more detail in the upcoming sections. As stated above, it helps us to obtain

the player's utilities for our game-theoretic model. In the context of prioritizing vulnerabilities to fix in MTD systems, we did not find any related work.

3.6 Concluding Remarks

In this chapter, we modeled a Moving Target Defense for web-applications in a formal game-theoretic framework. This modeling helped us to consider uncertainty over pre-defined attacker types and known real-world exploits present in the system in trying to come up with an efficient movement strategy for shifting among the defender's configurations C in the MTD. We note the use of (1) the National Vulnerability Database to mine relevant attacks and (2) the Common Vulnerability Scoring Service to come up with rewards of our game; we plan to leverage their use in the upcoming chapters. We adapted an existing Mixed-Integer Quadratic Program to fine-tune the Stackelberg strategy of our formulated game, accounting for the switching costs associated with movement.

Chapter 4

INFERRING MOVEMENT STRATEGIES AT STACKELBERG EQUILIBRIUM FOR MOVING TARGET INTRUSION DETECTION IN CLOUD NETWORKS

Table of Contents ◇ 1 ◇ 2 ◇ 3 ◇ 4 ◇ 5 ◇ 6 ◇ 7 ◇ 8 ◇ 9

C	Detection Surface Shifting in Cloud Networks
t	Constant/Fixed Time Period
M	[4.1] Stackelberg Strategy of a Normal-form Games with Large Defender Action Set
	[4.2] Stackelberg Strategy of a Markov Game effective against Multi-stage Attacks

---

Cyber adversaries often plan attacks that start with access over a public network interface and, via a slow-and-low multi-stage approach, end at a particular desired location in the defender's system. While defenders leverage measures such as placing Intrusion Detection Systems (IDS) to monitor an adversary's attack behavior, deploying all possible IDSs can have a negative impact on the system's performance. Hence, an idea to reduce the impact on performance is to place a subset of IDS systems at a time and move the detection surface continually.

In modeling this moving target approach as a game, the defender's strategy set needs to consider all possible subsets for placement of IDS systems resulting in a

combinatorial explosion of their action set $A^{\mathcal{D}}$. In Section 4.1, we address this issue by imposing a particular structure on the rewards of the game but assume that the attacker is capable of attacking any point of the cyber-system at any point in time.

In contrast, Section 4.2 assumes that an attacker can plan multi-stage attacks. Using the normal-form game-theoretic modeling discussed in Chapter 3 and Section 4.1 results in infinite number of attack strategies (where the attacker may consider loopy paths through the cyber system). To address this challenge, we model the problem as a Markov Game. While the Markovian and the fully-observable assumption impose certain restrictions, we show that our modeling helps up be far-more scalable in cloud-network scenarios.

4.1 Movement Strategies against Single-Stage Attacks

System Administrators, often, use Intrusion Detection Systems (IDS) to detect attack vectors in modern-day cyber-systems [82]. These IDS systems perform sophisticated operations– like signature-matching [83], anomaly detection [84, 85], machine learning [86, 87, 88] etc. – to investigate either live traffic on the wire (using Network-based IDS (NIDS) [89, 90]), or monitor resources on a machine (using Host-based IDS (HIDS) [91, 92]) to flag anomalous requests that might result in potential loss of confidentiality, integrity or availability. Cloud service providers, who host third-party applications on their platform, may encounter non-trivial challenges when it comes to deploying these IDS that can identify vulnerabilities present in their system due to operational constraints and deployment of applications that use legacy software [93]. The foremost among these challenges is the placement of IDS on all nodes of a large network, which results in reduced performance [94, 34] (also see Section 4.1.3).

Moreover, third party users of the cloud platform, due to privacy and security reasons, have constraints about sharing their data with the cloud provider [95].

Thus, given a cloud service provider's performance constraints and privacy constraints of customers, we look at the problem of placing a limited number of IDS systems in the various nodes of the cloud system. It is trivial to see that if we place IDS systems statically that only monitor certain attacks on specific nodes, an attacker (especially a *stealthy one*, i.e. one who resides inside a deployed system and can attack a node anywhere in the network, in contrast to having access to only hosts at the entry point) will eventually figure out our placement strategy [34]. At this point, a strategic attacker can always select attacks that circumvent the placed detection systems, thus passing through our cloud network undetected [96]. To address this, we design a Moving Target Defense (MTD) approach for dynamic placement of IDS systems on cloud systems.

The MTD placement mechanism for the cloud framework places both Network and Host-based IDS (abbreviated as NIDS and HIDS respectively). In this chapter, we will use a NIDS called `snort` [97] for detecting malicious behavior over the network and a HIDS known as `auditd` on the hosts of our cloud system. Further, we will assume is that NIDS is placed at the gateway of each tenant network and the HIDS is deployed on each Virtual Machine (VM). A dynamic switching (or MTD) strategy selectively turns `on/off` the different NIDS or HIDS systems; it can be used to monitor network packets using NIDS or the behavior of hosts using HIDS, and shift the detection surface at each round. The `on/off` commands sent out by a centralized entity, easily available in Software Defined Networking (SDN), hardly impact system performance at the time of switching. Hence, we do not consider switching costs in this chapter. The key contributions we make in this section are,

- We formulate the problem of placing limited IDS systems in a large cloud-based network using MTD as a two-player normal-form game between the defender and an attacker. Similar to the previous chapter, the Strong Stackelberg equilibrium of this game provided the defender with an optimal movement strategy that shifts the various IDS placements.

- We obtain the utility values of the players in this game by combining (1) the Common Vulnerability Scoring System (CVSS) that has been previously used to represent the impact of attacks on the defender's system [98] and (2) the centrality values of the nodes in which an IDS is deployed that lets us capture both the connectivity information and the impact on performance when an IDS is placed on that node [34].

- We design a scalable optimization problem to find the Stackelberg Equilibrium of our formulated game; it gracefully deals with the combinatorial explosion of the defender's pure-strategy set (subsection 4.1.2). Further, we allow an input parameter α that lets a defender balance between the security of the system and the impact on the performance of the system.

- We define the problem of finding the most critical vulnerability in a cloud environment with a strategic attacker and propose a method to solve it (Section 4.1.2).

- We demonstrate the effectiveness of our approach on a simulated example and compare it to several state-of-the-art strategies. We then provide experimental results in a small-scale cloud-based environment (subsection 4.2.3).

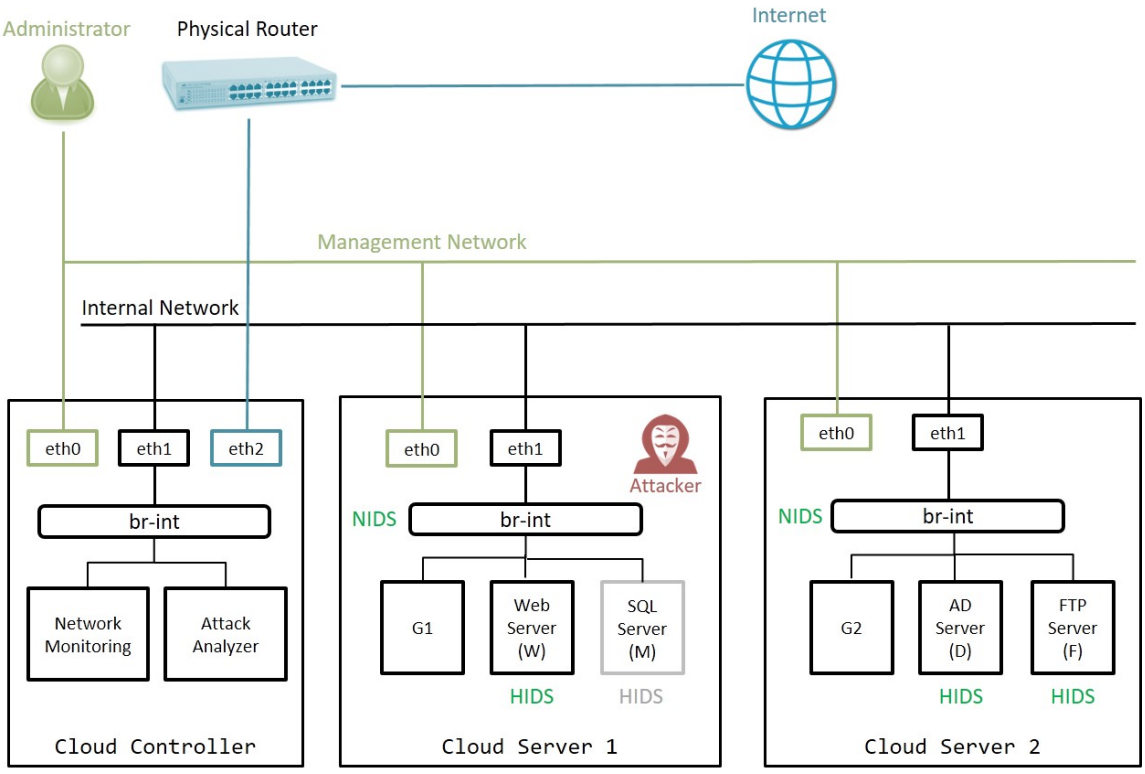


Figure 4.1: Defender's system on the enterprise cloud that an attacker wants to attack.

4.1.1 Game-Theoretic Modeling

In this section, we describe the various aspects of the game-theoretic model– the players and their action/pure-strategy sets using a small real-world scenario that we set up on an enterprise cloud (Figure 4.1). We then discuss how we can leverage (1) CVSS data and (2) the network topology information to obtain the rewards of the formulated game.

Threat Model In our attack model, we consider a multi-tenant cloud network. The controller node, shown in the Figure 4.1, is used for network management and orchestration. The network administrator (or the defender) utilizes a management

ID	VM	c_b	Vulnerability	CVE ID	IOC
a_1	G1	4	SSH Buffer Overflow	CVE-2016-6289	NIDS `sshAlert`
a_2	G2	7	`rlogin`	CVE-1999-0651	NIDS `rlogin`
a_3	W	0	Cross Side Scripting	CVE-2016-2163	HIDS `webAccess`
a_4	D	0	Weak Credentials	CVE-2001-0839	HIDS `fileIntegrity`
a_5	F	0	`vsftpd` backdoor	CVE-2015-1419	HIDS `ftpLogin`

Table 4.1: The different virtual machines (VMs) deployed on the defender's network, their betweenness centrality (c_b) in the graph, the known vulnerabilities in these nodes, and the corresponding Network/Host-based Intrusion Detection Systems (NIDS/HIDS) which can detect these attacks, also known as the Indicators of Compromise (IoC).

network to access controller nodes and cloud servers hosting VMs. We consider two agents– the defender $\mathcal{D}$, who is trying to deploy IDS and an (external or stealthy) attacker $\mathcal{A}$, who is trying to remain undetected while attacking the system. As a running example, we will use the scenario deployed by $\mathcal{D}$ shown in Figure 4.1. Furthermore, this system has a set of known vulnerabilities, that are yet to be fixed and as per our assumptions, known to both the agents $\mathcal{D}$ and $\mathcal{A}$.

We assume that the attacker $\mathcal{A}$ can be located either inside or outside the cloud network. The attacker's primary goal is to (1) compromise a VM using known vulnerabilities and (2) remain undetected while doing so. Since the attacker can utilize network probing to identify the OS and software versions, it will eventually get to know the vulnerabilities (CVEs) associated with the system, and can then systematically exploit these in order to obtain network access or elevated privileges. Furthermore, the attacker can only be detected when it attacks a vulnerability for which the

corresponding IDS is in place at the time of exploitation. For stealthy attackers [99], who have to spend a lot of cost and/or effort in gaining access to an internal node, the latter is of utmost importance.

Now given our system, we can extract the set A of all the n known vulnerabilities in our system (i.e. $n = (|A|)$). We choose the a_i IDs in the first column of Table 4.1 to represent an *attack* (and, also use a_i to denote the corresponding IDS that detects this attack). Thus, $n = 5$ and the set $A = \{a_1, a_2, a_3, a_4, a_5\}$. Note that this ID encodes a two-tuple ⟨MachineName, CVE-ID⟩; multiple attacks corresponding to a single machine, and similar vulnerabilities on different machines will each receive a unique ID.

The defender $\mathcal{D}$, as mentioned before, has a limited budget to place only $k(< n)$ IDS mechanisms due to resource constraints. Also, we assume that, due to privacy constraints, $\mathcal{D}$ cannot place an IDS mechanism on the 'SQL Server (M)' (shown in Figure 4.1). Thus, in our model, we disregard any vulnerabilities present on this node. (Note that although our system can detect a class of vulnerabilities that trigger NIDS alarms on the network interface G1 when they affect M, we exclude such vulnerabilities from our example). Now, $\mathcal{D}$ has $\binom{n}{k}$ ways in which it can deploy the k IDSs. This is the action set of $\mathcal{D}$. Formally, the defender's action set is denoted by the set $A^{\mathcal{D}} = A_k = \{S \in A : |S| = k\}$. In the running example, we will assume that $k = 2$. Thus, the defender's action set is:

$$\{(a_1, a_2), (a_1, a_3), (a_1, a_4), (a_1, a_5), (a_2, a_3), (a_2, a_4), (a_2, a_5), (a_3, a_4), (a_3, a_5), (a_4, a_5)\}$$

We assume a strong adversary who either knows or can find out all the attacks in our system. Thus, the action set of the attacker is $A^{\mathcal{A}} = A = \{a_1, a_2, a_3, a_4, a_5\}$.

In game theory, this action set is often referred to as the set of pure strategies, where each action (either a placement strategy or an attack) is a pure strategy (for

$\mathcal{D}$ or $\mathcal{A}$ respectively). As stated earlier, if a defender chooses a pure strategy, i.e, any one out of the ten pure strategies shown, to deploy k IDS systems, the attacker, with reconnaissance on its side, will eventually figure out $\mathcal{D}$'s strategy and start choosing attacks that do not trigger these alarms. In order to address this limitation, the defender can play a mixed strategy, i.e. have a probability associated with playing each pure strategy and at the start of each round pick one by randomly sampling a pure strategy from the set of pure strategies. Note that this is similar to applying the concept of Moving Target Defense where the defender chooses to switch randomly among the different deployment configurations (i.e. by choosing one of the ten IDS placements in our case) at the start of each time period.

Common Vulnerability Scoring System (CVSS) As already discussed in Chapter 3, the CVSS metric provides two quantitative scores for each CVE present in our system–(1) the Impact Score (IS) that represents the effect a particular attack has on the Confidentiality, Integrity, and Availability of a system and (2) the Exploitability Score (ES), which encodes the complexity of actually exploiting a particular vulnerability. The system defines a way to combine both of these scores to calculate a third score, known as the Base Score (BS) that tries to consider both the impact of an attack *vs.* the difficulty in exploiting it.

The CVSS scores thus leverages the knowledge of cybersecurity experts across the globe to provide a numerical value corresponding to each (known) vulnerability that reflects its severity and expertise necessary to exploit it. We, inspired by other research work before us [19, 50, 20], use the CVSS to calibrate the reward values of our game.

Stackelberg Games

Having defined the players and their action (or pure strategy) sets, there are additional real-world aspects that we want to incorporate in the formulation of our game. One such aspect is that the defender, who hosts the system that an attacker attacks, plays first. To accurately model this scenario, we use the concept of Stackelberg games in which one player ($\mathcal{D}$) acts before the other player ($\mathcal{A}$) plays and find the Stackelberg Equilibrium of these games, in which the leader's ($\mathcal{D}$) strategy is contingent upon the fact that the follower ($\mathcal{A}$) can observe $\mathcal{D}$'s strategy and play accordingly. Thus, in this adversarial leader-follower game, $\mathcal{D}$ can simulate $\mathcal{A}$ in their mind and decide on a mixed strategy that gives it the highest utility keeping in mind (that a rational) $\mathcal{A}$ will choose the best action ($\in A^{\mathcal{A}}$), i.e. the action that maximizes $\mathcal{A}$'s reward, in response.

Utility Modeling

Having designed the action sets of both the players, we can now specify the utilities for both the players when each of them commits to a pure strategy. Given $|A^{\mathcal{D}}| = \binom{n}{k}$ and $|A^{\mathcal{A}}| = \binom{n}{k}$, to enumerate all the utility values for our normal-form game we would have to specify $2 \cdot \binom{n}{k} \cdot n$ values corresponding to the reward values for each of the players $\mathcal{D}$ and $\mathcal{A}$. With this general reward structure, finding the mixed-strategy Stackelberg equilibrium of this game would be computationally inefficient, specifically $O(\binom{n}{k})$ [62]. Thus, we now resort to a more restrictive reward structure that lets us efficiently compute the equilibrium strategy while being able to capture all the aspects of our problem.

For each attack $a \in A$, if $\mathcal{D}$ places an IDS to detect it, we will say that $\mathcal{D}$ *covers* it. Otherwise, we say that $\mathcal{A}$ is left *uncovered*. Since the defender can allocate only IDS

resources to cover k elements in $\mathcal{A}$, the remaining $n-k$ attacks will remain uncovered at any point in time. We will now decompose the reward structure of this game and define four types of utility values corresponding to each attack $a \in A$.

$$\langle R^{\mathcal{D}}_{c,a}, R^{\mathcal{D}}_{u,a}, R^{\mathcal{A}}_{c,a}, R^{\mathcal{A}}_{u,a} \rangle$$

where $R^{\mathcal{D}}_{c,a}$ and $R^{\mathcal{D}}_{u,a}$ denotes the utilities that a defender gets for covering and not covering an attack $\mathcal{A}$ respectively. Similarly, $R^{\mathcal{A}}_{c,a}$ and $R^{\mathcal{A}}_{u,a}$ represent the utility an attacker gets when they use an attack $\mathcal{A}$ that is covered (and thus gets detected) or not covered (and thus avoids detection) respectively. The values for these symbols are obtained by leveraging the knowledge of security experts as encoded in the Common Vulnerabilities Scoring System (CVSS) [100] and the realistic costs of deploying IDSs. For each attack a_i in the set of attack actions $\mathcal{A}$, we will represent these scores as IS_{a_i}, ES_{a_i} and BS_{a_i} using CVSS metrics, previously discussed in Sec. 4.1.1.

Cost of deploying IDS. We denote the cost of deploying an IDS corresponding to an attack $a \in A$ as $\hat{c}_a$. For our example, we assume the cost of deploying an IDS (shown in the IOC column of table 4.1) to be proportional to the betweenness centrality of the VMs on which the IDS is deployed because a VM with high betweenness centrality will affect the latency of routing packets or the latency of processing a request. Also, the centrality values are normalized in the interval $[0, 10]$ to be comparable to the CVSS metrics IS_a, ES_a and BS_a as discussed in Sec. 4.1.1. Note that the model proposed allows another user to define $\hat{c}_a$ in a different way.

We now leverage these defined metrics to design the following rewards for the four utilities associated with each attack $\mathcal{A}$ present in our system,

$$R^{\mathcal{D}}_{c,a} = -1 * \hat{c}_a \quad , \quad R^{\mathcal{D}}_{u,a} = -1 * IS_a$$

$$R^{\mathcal{A}}_{c,a} = -1 * ES_a \quad , \quad R^{\mathcal{A}}_{u,a} = +1 * BS_a$$

We now provide the rationale for modeling the rewards in this particular manner. The value of $R^{\mathcal{D}}_{c,a}$ is negative since even if it detected an attack, it incurred a cost in order to detect it, and moreover, there is no extra positive reward given to $\mathcal{D}$ for protecting their system, which is supposed to be the primary functionality. When $\mathcal{D}$ does not place an IDS for detecting the attack $\mathcal{A}$, it incurs a negative utility ($R^{\mathcal{D}}_{u,a}$) equal to IS_a if the attacker uses attack $\mathcal{A}$.

For the attacker $\mathcal{A}$, if it chooses an attack action $\mathcal{A}$ which the defender covers (i.e. can detect), it gets a negative utility $R^{\mathcal{A}}_{c,a}$ proportional to the time and cost it had to invest in doing it, which is (somewhat) measured by ES. Also, as $\mathcal{A}$ gains nothing by doing this attack (since the defender can deploy a countermeasure on detection [24]), no positive value is added to it. Lastly, when the attacker uses an attack for which the defender has not placed an IDS, we give a positive utility that (conceptually) adds the IS and subtracts the cost (ES) of performing the attack. Since BS already captures this trade-off, we use it directly.

4.1.2 Solution Concepts

We need to solve for the Stackelberg Equilibrium of our game to obtain probability values for each configuration mentioned in A_k, where $A_k \subset A$ such that $|A_k| = k$. Unfortunately, since there are $\binom{n}{k}$ such probabilities (corresponding to each element in A_k), solving for all these variables at once will not yield an efficient solution. Instead, we will solve for the probabilities p_a which represents the probability that a certain attack $a \in A$ is covered by an IDS in a round.

To that extent, we first describe a method that can help in generating the marginal strategies for the defender by solving n ($= |A^{\mathcal{A}}|$) Linear Programs. Note that the solution can be found in polynomial time in our case because of the particular reward

structure our game has. Then, we shall propose an efficient Mixed-Integer Quadratic Program (MIQP) method based on this method that helps us to obtain the same marginal strategy, but by solving just one optimization problem. We show that, although this formulation in the general case is known to be computationally hard, in our case can be solved by using a branch-and-cut mechanism in polynomial time.

Multiple LP method

Let us first define the notion of a *token* that can be allocated to an attack. When a token is to an attack a, it implies that $\mathcal{D}$ has deployed an IDS that can detect this particular attack. Let T denote the set of k tokens that the defender $\mathcal{D}$ can allocate to cover k of the n attacks. Now, let the variables p_a represent the probability with which an attack a is covered by one of the k tokens and $p_{a,t}$ represent the probability with which a particular attack a is covered by a particular token $t \in T$. Having defined the probabilities p_a, the defender's expected utility for deploying an IDS to detect a particular attack a^* should be $R^{\mathcal{D}}_{u,a^*} * (1 - p_{a^*}) + R^{\mathcal{D}}_{c,a^*} * p_{a^*}$ [101, 102]. Note that, for our scenario, this does not capture the cost $\mathcal{D}$ incurs in deploying the other $k-1$ IDS mechanisms. Thus, we modify the defender's utility to $R^{\mathcal{D}}_{u,a^*} * (1 - p_{a^*}) + \frac{1}{k} \sum_{a \in A} R^{\mathcal{D}}_{c,a} * p_a$, where the second term denotes the average cost for a particular deployment configuration.

On the other hand, we can simply define the attacker's expected utility for using a particular attack a as $R^{\mathcal{A}}_{c,a} * p_a + R^{\mathcal{A}}_{u,a} * (1 - p_a)$. We now present the optimization problem that maximizes the defender's objective function and the attacker's utility given that an attacker chooses to use the attack a^*.

$$\begin{aligned} \max \quad & \alpha \cdot \frac{1}{k} \sum_{a \in A} R^{\mathcal{D}}_{c,a} p_a \quad + \quad (1-\alpha) \cdot R^{\mathcal{D}}_{u,a^*}(1 - p_{a^*}) \\ s.t. \quad & p_a \quad \in \quad [0,1] \quad \forall \; a \in A \end{aligned} \tag{4.1}$$

$$
\begin{aligned}
p_{t,a} &\in [0,1] \;\; \forall\, a \in A, t \in T \\
\sum_{a \in A} p_{t,a} &= 1 \;\; \forall\, t \in T \\
\sum_{t \in T} p_{t,a} &= p_a \;\; \forall\, a \in A \\
R^{\mathcal{A}}_{c,a} p_a + R^{\mathcal{A}}_{u,a}(1 - p_a) &\leq R^{\mathcal{A}}_{c,a^*} p_{a^*} + R^{\mathcal{A}}_{u,a^*}(1 - p_{a^*})
\end{aligned}
$$

where α is an input parameter that allows the defender to trade the performance of the system with respect to the security of the system (and vice versa). In the extreme case when $\alpha = 0$, the defender optimizes only for security and completely ignores the fact that deploying k IDSs might affect the performance of the system; as shown in subsection 4.2.3, $\mathcal{D}$ ends up moving between a larger number of IDS deployment configurations. On the other hand, when $\alpha = 1$, the defender optimizes for performance, hardly placing an IDS on systems that affect performance even when such a choice is detrimental to security. We discuss the effects of selecting various α-s in subsection 4.2.3.

Before we dive into what the constraints mean, note that this is a Linear Program (LP) and thus, can be solved in polynomial time. The first two sets of constraints ensure that the optimization variables p_a and $p_{t,a}$ are valid probabilities. The third set of constraints ensures that all the tokens are utilized in covering the different attacks in $\mathcal{A}$. The equality of this constraint is possible in our case since (1) all our tokens are homogeneous, i.e. any token $t \in T$ can be used to cover any attack $a \in A$ and (2) the number of tokens k ($= |T|$) is less than the number of attacks n ($= |A^{\mathcal{A}}|$). Thus, we prune away solutions that do not fully utilize all the tokens. The fourth set of constraints ensure that the probabilities of allocating various tokens to cover an attack $\mathcal{A}$ add up to the probability that $\mathcal{A}$ is covered. The final set of constraints ensure that the attacker selecting a^* maximizes their utility. Lastly, note that given the values of $p_{t,a}$ one can easily obtain p_a using the fourth set of constraints.

To obtain the (globally) optimal solution (and thus find the optimal marginal strategy) for the defender, we can iterate over all the n attack choices made by the attacker and pick the solution that maximizes $\mathcal{D}$'s utility. Note that, here we enforce the attacker to select a pure strategy as opposed to a mixed strategy. This is not a limitation since for any mixed strategy the attacker can pick in this Stackelberg Game, there always exists a pure strategy in support of it [103].

As the number of VMs and vulnerabilities, i.e., n, increase, this solution method needs to solve a large number of LPs. Thus, we now propose an efficient MIQP that finds the solution in one go and provides an efficient branch-and-cut algorithm for solving it in polynomial time.

Compiling Multiple LPs into an Efficient Mixed-Integer Quadratic Program (MIQP)

Now, we first introduce n binary switch variables, one for each attack $a \in A$ and represent it as w_a. When the attacker exploits vulnerability $\mathcal{A}$ (i.e. uses the attack action $\mathcal{A}$), $w_a = 1$. Otherwise, $w_a = 0$. We now propose the following optimization problem,

$$
\begin{aligned}
\max \quad \alpha \cdot \frac{1}{k} \sum_{a \in A} R^{\mathcal{D}}_{c,a} p_a \quad &+ \quad (1-\alpha) \cdot w_a * R^{\mathcal{D}}_{u,a}(1-p_a) \qquad (4.2)\\
s.t. \quad w_a \quad &\in \quad \{0,1\} \quad \forall\ a \in A \\
p_a \quad &\in \quad [0,1] \quad \forall\ a \in A \\
p_{t,a} \quad &\in \quad [0,1] \quad \forall\ a \in A, t \in T \\
\sum_{a \in A} w_a \quad &= \quad 1 \\
\sum_{a \in A} p_{t,a} \quad &= \quad 1 \quad \forall\ t \in T
\end{aligned}
$$

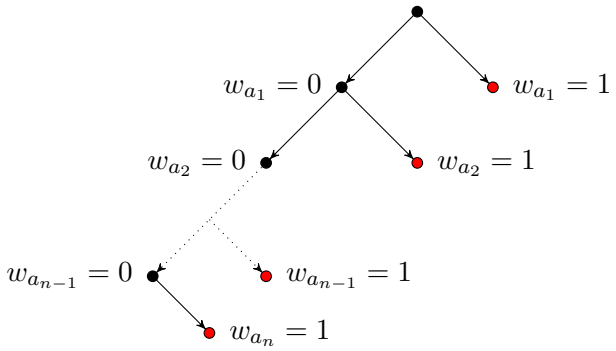


Figure 4.2: The branching tree for the proposed MIQP.

$$\begin{aligned} \sum_{t \in T} p_{t,a} &= p_a \quad \forall\ a \in A \\ 0 \leq v_a - (R^{\mathcal{A}}_{c,a} p_a + R^{\mathcal{A}}_{u,a}(1 - p_a)) &\leq (1 - w_a) * M \quad \forall\ a \in A \end{aligned}$$

where M represents a large number with respect to the maximum reward the attacker can get, i.e. $M >> 10$, and v_a is the utility value of the attacker at equilibrium. The first constraint ensures that the switch variables are binary. The fourth constraint enforces the attacker to select a pure strategy since the switch variable corresponding to only one attack can be turned on in a feasible solution. As mentioned in the previous section, this is not a limiting assumption. Lastly, the final set of constraints encodes the complementary slackness condition of the attacker's utility maximization problem [103].

As the defender plays first, it can reason about the attacker picking each attack and select the strategy which gives $\mathcal{D}$ the maximum reward. If the attacker responds to the defender's strategy with attack a^*, then $w_{a^*} = 1$. In that case, the RHS of the last constraint (with a^*) becomes zero and along with the LHS, equality holds. Thus, v_{a^*} is $\mathcal{A}$'s utility value. For all the other attacks $a(\neq a^*)$ that were not selected by $\mathcal{A}$, both the inequalities can be trivially satisfied (as M is a large number) by selecting an appropriate value for v_a.

Theorem 4.1.1. *MIQP defined in Equation 4.5 produces the same solution as the set of LPs described in equation Equation 4.1.*

Proof. Let us say that when attacker selects an attack a_1, the defender gets the highest utility as per Equation 4.1. Now, let us say that Equation 4.5 decides that the defender's utility is strictly better when attacker selects any another attack $a_2(\neq a_1)$, and thus, $w_{a_2} = 1$. Notice that if this is true, then the objective function value of LP when $a^* = a_2$ is strictly greater than the objective function value of the LP with $a^* = a_1$. But that is a contradiction. Hence, the MIQP defined in Equation 4.5 must select a_1 for the attacker.

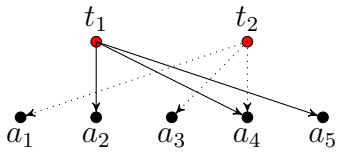
Similarly, we can prove the other way–that a solution that is optimal for the MIQP (Equation 4.5) is also optimal for the LP case. □

Theorem 4.1.2. *MIQP defined in Equation 4.5 can be solved in polynomial time with the branch-and-cut method.*

Proof. To prove this, we first represent the branch-and-cut tree for our MIQP in Figure 4.2. In that, notice that the right children (shown in red) correspond to an LP problem (similar to the one defined in Equation 4.1) where only a particular attack a_i is selected ($w_{a_i} = 1$) and other attacks are not used by the attacker. Since no children of any right child (red node) can generate another solution, the search tree below them can be pruned away. Now, the tree can have at most $n - 1$ left children which correspond to at most n right children, which in turn corresponds to at most n LP problems that need to be solved. Since each LP can be solved in polynomial time and we will solve no more than n LPs, this MIQP can be solved in polynomial time. □

Attack	a_1	a_2	a_3	a_4	a_5
$R^{\mathcal{D}}_{c,a}$	-5.7	-10.0	0.0	0.0	0.0
$R^{\mathcal{D}}_{u,a}$	-6.4	-6.4	-2.9	-6.4	-2.9
$R^{\mathcal{A}}_{c,a}$	-8.6	-10	-8.6	-10	-10
$R^{\mathcal{A}}_{u,a}$	6.8	7.5	4.3	7.5	5.0

Table 4.2: Player utilities for each vulnerability depending on whether an IDS is deployed (or not) to detect the attacks that exploit it.



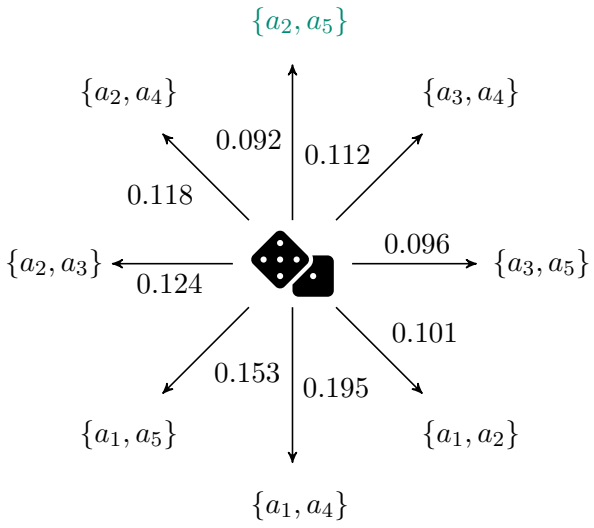
	a_1	a_2	a_3	a_4	a_5
t_1	0	0.44	0	0.22	0.34
t_2	0.45	0	0.34	0.21	0

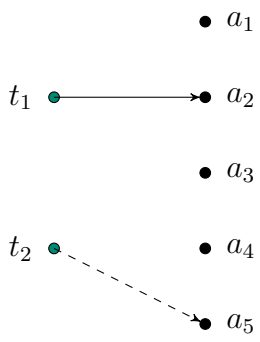
Table 4.3: Probability of allocating a *token* (that indicates whether the corresponding IDS should be deployed) for detecting each attack.

Obtaining Implementable Strategies

Although we have obtained the values p_a and $p_{t,a}$, there are no guarantees that we will be able to convert these marginal probabilities into $\binom{n}{k}$ probability values that correspond to a defender's deployment strategies, i.e. one that can be implemented in practice. In order to convert these into *implementable strategies*, we use the general version of the Birkhoff Von-Neumann Theorem as stated in [102]. We state this here for completeness.

Birkhoff Von-Neumann Theorem. *Consider an $k \times n$ matrix P with real numbers $p_{t,a} \in [0,1]$, such that for each $1 \leq t \leq k$, $\sum_{a=1}^{n} p_{t,a} \leq 1$, and for each $1 \leq a \leq n$, $\sum_{t=1}^{k} p_{t,a} \leq 1$. Then, there exist matrices $P^1, P^2, \ldots, P^q$ and weights $w^1, w^2, \ldots, w^q \in (0,1]$, such that (1) $\sum_{x=1}^{q} w^x = 1$; (2) $\sum_{x=1}^{q} w^x P^x = M$; (3) for each $1 \leq x \leq q$, the elements of M^x are $p^x_{t,a} \in \{0,1\}$ and (4) for each $1 \leq x \leq q$, we have for each $1 \leq t \leq k$, $\sum_{a=1}^{n} p^x_{t,a} \leq 1$ and for each $1 \leq a \leq n$, $\sum_{t=1}^{k} p^x_{t,a} \leq 1$.*





	a_1	a_2	a_3	a_4	a_5
t_1	0	1	0	0	0
t_2	0	0	0	0	1

a_1	a_2	a_3	a_4	a_5
0	1	0	0	1

Figure 4.3: Optimal mixed strategy of the defender for our scenario when $\alpha = 0.1$. The probability values for picking up one of the eight IDS placements at the start of each round are written on the edges. For the strategy $\{a_2, a_5\}$, the allocation matrix is shown on the right.

This theorem guarantees that given the probability matrix $p_{t,a}$, we can always obtain the probabilities of the $\binom{n}{k}$ implementable strategies. The third and fourth equalities in the optimization problem in Equation 4.1 ensure that the constraint structure imposed on P is a *bi-hierarchy*, which authors in [104] show as a sufficient condition for any marginal probability matrix P to be *implementable*.

For our example, assuming that the cost associated with deploying each IDS on a certain VM is a function of the latency it creates. Furthermore, since VMs that are

responsible for communication between other VMs would impact the latency the most when an IDS is placed on it. Thus, we assume time impact on the overall latency of the system is equal to the normalized and scaled betweenness centrality of the nodes in our network ($\in [0, 10]$). With that, the utility values for the attacker and defender are shown in Table 4.2. We first use these values to solve for the optimal marginal strategy (shown in Table 4.3) using the MIQP described in Equation 4.5. We then use Theorem 1 to obtain the mixed strategies that the defender can actually use to deploy the IDS systems (shown in Figure 4.3).

Identifying the Most Critical Vulnerability

In real-world scenarios, system administrators, who have a list of known vulnerabilities it should address, have limited developer resources to fix all of the known CVEs in their system at once. Thus, the question of which vulnerability they should fix in order to improve the security of the system is a critical one. In our case, since (1) the rewards of the formulated game are not zero-sum and (2) the defender wants to balance a multi-objective function (that tries to balance the security and usability metrics), figuring out the (critical) vulnerability that $\mathcal{D}$ needs to fix become even more difficult.

Given that we can find the utilities for the defender using Equation 4.5, we can ask the question *which attack $\mathcal{A}$ when removed would produce the maximum utility for $\mathcal{D}$?* A simple algorithm would be to iterate over all the attacks, removing them one by one, reformulating the MIQP, and selecting the attack that maximizes the defender's utility when removed. We describe this idea formally in Algorithm 1 and use it to find the most critical vulnerability of our system. [1] The utilities obtained

[1]Note that if we follow this method, we would need to solve $\binom{n}{f}$ MILPs (or $\binom{n}{f} \cdot n$ LPs) if the defender has resources to fix f vulnerabilities at one go.

Algorithm 1 Algorithm to find the most critical vulnerability in the Defender's system that upon patching results in the highest defender utility.

procedure GIVEN UTILITY MATRIX,
OUTPUT(a^* – the most critical vulnerability that results in the highest defender utility when fixed)
    max_def_util $\leftarrow -\infty$
    $a^* \leftarrow None$
    while $a \in A$ **do**
        $A' \leftarrow A \setminus a$
        obj_val, $\leftarrow$ solve MIQP in Equation 4.5 with action set A'
        if obj_val $>$ max_def_util **then**
            max_def_util $\leftarrow$ obj_val
            $a^* \leftarrow a$
        end if
    end while
    return a^*
end procedure

by removing one vulnerability at a time are shown below (for $\alpha = 0.1$).

$$\langle a_1 : -1.90; a_2 : -1.70; a_3 : -2.30; a_4 : -2.23; a_5 : -2.27 \rangle$$

Thus, in our system, a_2 is the most critical vulnerability since fixing a_2 will result in the highest (gain in) defender's utility.

Strategy	a_1	a_2	a_3	a_4	a_5
URS	0.4	0.4	0.4	0.4	0.4
DPS	1	1	0	0	0
CBS	0.52	0.73	0.25	0.25	0.25

Figure 4.4: The marginal probabilities, associated with different movement strategies, with which an IDS is placed to detect an attack.

4.1.3 Experimental Results

We present the results of two different experiments– (1) comparison of our placement strategy (Fig. 3) with existing approaches, and (2) implementation of the Stackelberg Game Strategy (SGS) on a small-scale cloud network instance.

Comparison with Existing Strategies

In this section, we compare our approach to three other MTD strategies in the context of our running example where $n = 5$ and $k = 2$–

(1) Deterministic Pure Strategy (DPS). This strategy selects a single pure strategy out of the $\binom{5}{2}$ placement strategies. As per work by [94], for DPS, we place IDS to detect a_1 and a_2 (since G1 and G2 are the most critical VMs), which are on the critical paths for any attack flow. Note that, in the context of a stealthy attacker who can exploit any vulnerability in the system, the definition of a critical node, on which an IDS can be deployed, is not clear. Thus, DPS has an inherent disadvantage when compared to MTD strategies, which we now describe.

(2) Uniform Random Strategy (URS). In this case, we select each of the $\binom{5}{2}$ placements or pure strategies with an equal probability of 0.1. In this case, each attack $\mathcal{A}$ is covered in four (out of the ten) pure strategies since having placed an IDS (or token which denotes an IDS was placed) for $\mathcal{A}$, there are $\binom{4}{1} = 4$ ways of placing the other token. Thus, the marginal probabilities are $0.1 * 4 = 0.4$.

(3) Centrality Based Strategy (CBS). This strategy, motivated in the work by [34], has previously been shown to be effective for detecting stealthy bot-nets when PageRank is used as a centrality measure. Since our network is an undirected graph, we use the betweenness centrality measure for evaluation. Since only two of our nodes (G1 and G2) have non-zero values for betweenness centrality, we switch between seven of the ten configurations– three in which only a_1 is covered, three in which only a_2 is covered and one in which both a_1 and a_2 are covered. Since $G1$, on which a_1 is present has a lower centrality value in comparison to $G2$, on which a_2 is present, the first three configurations are less likely than the next three. The last configuration, in which both a_1 and a_2 are covered, is the most likely configuration. The marginal probabilities for covering each attack in the system, as per this strategy, are shown in Figure 4.4.

Effectiveness of Our Approach We plot the defender's utility value for our approach and compare it to all the other approaches. The results are shown in Figure 4.11. When adversaries are strategic, i.e. can reason about defender strategies and act rationally to maximize their utility, our method clearly dominates the other methods (see the plots for CBS(min), URS(min) and DPS).

On the other hand, if the attacker is irrational, i.e., selects attacks that do not maximize their profit, Stackelberg Equilibrium may not always be the best strategy.

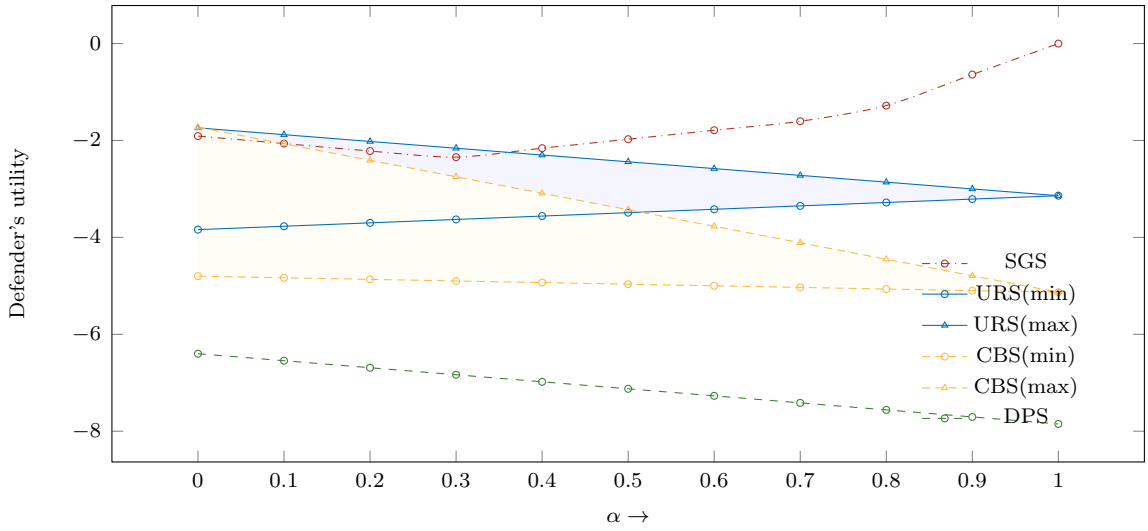


Figure 4.5: Defender's utility for the various movement strategies as the parameter α, which represents the security-usability trade-off, varies from zero to one.

We plot the best case for the other MTD strategies (see URS(max) and CBS(max)) and it turns out that only URS is a little better when $\alpha \in (0, 0.37]$. In this range, our algorithm selects nodes with high centrality measures to improve security in the case of a strategic attacker. This increases the deployment cost and reduces the multi-objective function value, letting URS dominate. CBS on the other hand with no information about the known attacks or performance costs, switches only among the useless and performance expensive configurations, being strictly dominated by SGS. Note that none of the mechanisms we compare against adapt to the security and performance trade-off that is important to the defender. Thus, as the value of α changes, the marginal probabilities for selecting nodes using CBS, URS, or DPS remain constant, resulting in straight-line plots. On the other hand, SGS, our intelligent switching mechanism, solves the multi-objective optimization when coming up with its mixed strategy.

When α is low (i.e. $\in [0, 0.29]$), our method switches among eight out of the ten pure strategies. As α increases further and the costs start to matter, it places IDS

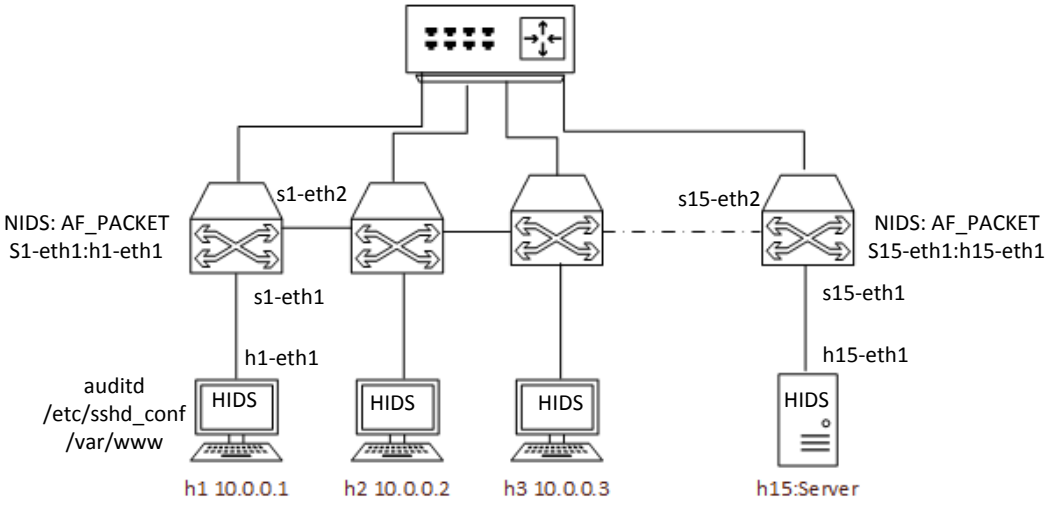


Figure 4.6: Testing bandwidth on a flat network with 15 virtual machines and various Network and Host Intrusion Detection Systems.

systems more on nodes that impact performance the least. Beyond a certain value (when $\alpha > 0.76$) it realizes that the cost of placing IDS on G1 and G2 (for detecting $a1$ and $a2$) are extremely high on the performance of the system and sticks to only (three) strategies where neither G1 nor G2 is covered.

Testing on a Real-world Cloud Network

The setup comprised of 15 VMs and 42 CVEs distributed uniformly on a flat network 10.0.0.0/24, as shown in the Figure 4.6. In this experiment, we will measure the throughput for the server (10.0.0.15) hosting an ssh application on port 5002 as the number of IDS systems placed increases. We now describe the different NIDS and HIDS agents pre-configured on the system with the known attack signatures to detect the intrusion attempts.

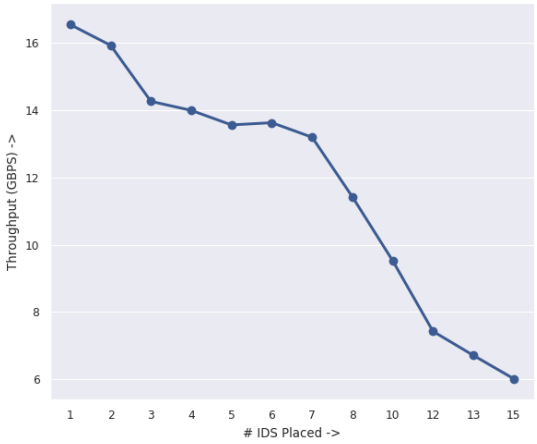


Figure 4.7: Change in throughput of the flat network as the number of NIDS and HIDS deployed increases.

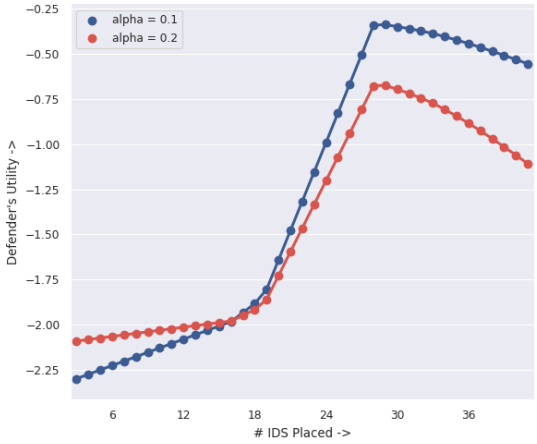


Figure 4.8: Change in defender's utility value as the number of NIDS and HIDS deployed increases.

Network-based IDS Snort [97] was configured to run in IDS (intrusion-detection) as well as IPS (intrusion-prevention) mode. For instance, the attack signature below checks the payload for shell-code targeting remote buffer overflow vulnerability on ssh service running on port 5022.

```
alert TCP any any -> 10.0.0.15 5002 (msg:"EXPLOIT ssh remote
  overflow"; content:"/bin/sh"; reference:Bugtraq,2347;reference:
  cve,2008-5161; sid:1324; rev:6;)
```

The AF packet, which is an IPS configuration, creates a bridge between inspected interfaces (e.g., h1-eth1:s1-eth1). This leads to increased packet processing latency since each packet on a particular bridge is inspected against all traffic patterns which are part of signatures.

Host-based IDS auditd [105] was configured to monitor file integrity of configuration files such as /etc/sshd_conf and binary files for vulnerable services present on

the network. A daemon was configured on each inspected host to generate an alert if there is a change in the hash value of inspected files.

The goal of this experiment was to measure the impact of the HIDS/NIDS deployment on the throughput of the service being accessed by normal users. We show that as $\mathcal{D}$ places more IDSs (1 to 15), we observe a substantial drop in the throughput of the system from 18 Gbps to 6 Gbps (see Figure 4.7). This shows that the deployment of IDS without considering the impact on network latency can affect the Quality of Service (QoS) for legitimate users in a cloud network.

In Figure 4.8, we vary the number of IDS systems placed in the system and see how the defender utilities vary. Initially, as the number of IDS increases from 2 to 17, the defender's utility increases at a slow rate since there are too few IDS systems to detect attacks on all the 42 vulnerabilities. As the number of IDS systems is increased beyond 18, the defender's utility starts to increase substantially in each step. At this point, if the attacker does not pick their attack strategically, it is detected with high probability. However, the placement of IDSs beyond a certain threshold– 29 as shown in the Figure 4.8– results in a substantial decrease in throughput, outweighing the benefits of security provided by IDS.

We will now consider an even more pragmatic threat model where an attacker can plan and execute a multi-step attack. We allow a defender's sensor model and an attacker's attack execution to be imperfect. Under these conditions, we will show how one can find the optimal movement strategy.

4.2 Movement Strategies against Multi-Stage Attacks

In the case of real-world cloud-systems, the solution discussed in Section 4.1 leads to three problems. First, only single-step attacks are considered in the game-theoretic modeling which leads to sub-optimal strategies because such models fail to account for multi-stage attack behavior. For example, strategies generated by prior work may prioritize detecting a high-impact attack on a web-server more than a low-impact attack on a path that leads an attack on a storage server, which when exploited may have major consequences. Second, the threat model assumes that an attacker can launch an attack from any node in the cloud network. This is too strong an assumption, leading to sub-optimal placement strategies (that are too conservative) in real-world settings. Third, existing methods can come up with placement strategies that allocate multiple detection systems on a sub-net while another sub-net is not monitored. This results in a steep degradation of performance for some customers. To address these challenges, we need a suitable method for modeling multi-stage attacks. Unfortunately, capturing all possible attack paths can lead to a combinatorial explosion of the pure-strategy set $A^{\mathcal{A}}$. Thus, we use *Markov Games* to model such interactions.

Specifically, we try to address these problems by modeling the cloud system as a General-Sum Markov Game. We use particular nodes of our system's *attack graph* to represent the states of our game. Similar to the works discussed previously, the attacker's actions are modeled after real-world attacks based on the Common Vulnerabilities and Exploits (CVEs) described in the National Vulnerability Database (NVD) [106] while the defender's actions correspond to the placement of detection systems that can detect the attacks. The utility values for each player in this game are obtained by leveraging (1) the CVSS metrics corresponding to known attack actions (similar to previous work), and (2) cloud designer's quantification of how the

placement of a detection system impacts the performance. These help us come up with defense strategies that take into account the long-term impacts of a multi-stage attack while restricting the defender to pick a limited number of monitoring actions in each part of the cloud. The latter constraints ensure that the performance impact on a customer, due to the placement of detection measures, is limited.

The popular notion of using min-max equilibrium for Markov Games [107] results in an optimal strategy for the players in zero-sum games but, may yield sub-optimal policies if the rewards have a general-sum structure. Given that an attacker is (eventually) aware of the defender's placement strategy (in each state of our Markov game) in our threat model, we consider the notion of Strong Stackelberg Equilibrium of this game. In scenarios where this assumption becomes too strong, we show that the Stackelberg Equilibrium of our general-sum game, depending on the problem structure, is a subset of Nash Equilibrium. Thus, it still results in good movement strategies. In summary, we make the following contributions.

- We model the multi-stage attack scenarios, which are typically employed in launching Advanced Persistent Threats (APTs) campaigns against high-value targets, as a general-sum Markov Game. The cost-benefit analysis based on the two-player game provides strategies for placing detection systems in a cloud network.

- We leverage the attack-graph modeling of cloud networks, the Common Vulnerabilities and Exposures (CVEs) present in the National Vulnerability Database and the Common Vulnerability Scoring Service (CVSS) to design the states, the actions and utility values of our game. In addition, we consider metrics discussed in the context of cloud-systems to (1) model the uncertainty of an at-

tack's success and (2) leverage heuristic measures that model the performance impact of placing detection mechanisms on the cloud.

- Our framework considers a threat model where the attacker can infer the defender's detection strategy. Therefore, we design a dynamic programming solution to find the Stackelberg equilibrium of the Markov Game. If an attacker does not have information about the defender's strategy, we show that the Stackelberg equilibrium of the general-sum Markov Game is a subset of Nash Equilibrium given a set of properties are satisfied (similar to prior work in normal-form games [108]). In order to showcase the effectiveness of our approach, we analyze a synthetic and a real-world cloud system.

4.2.1 Background

In this section, we first introduce the reader to the notion of real-world vulnerabilities and exploits present in a cloud system that we will use throughout our paper. Second, we describe the threat model for our cloud scenario. Lastly, we describe the notion of Attack Graphs (AG) followed by a brief description of Markov games and some well-known algorithms used to find the optimal policy or strategy for each player. We will use the example attack scenario for cloud networks shown in Figure 4.9 as a running example in our discussion.

Vulnerabilities and Exploits

Software security is defined in terms of three characteristics - Confidentiality, Integration, and Availability [71]. Thus, in a broad sense, a vulnerability (that can be attacked or exploited) for a cloud system can be defined as a security flaw in a software service hosted over a given port. When exploited by a malicious attacker, it can

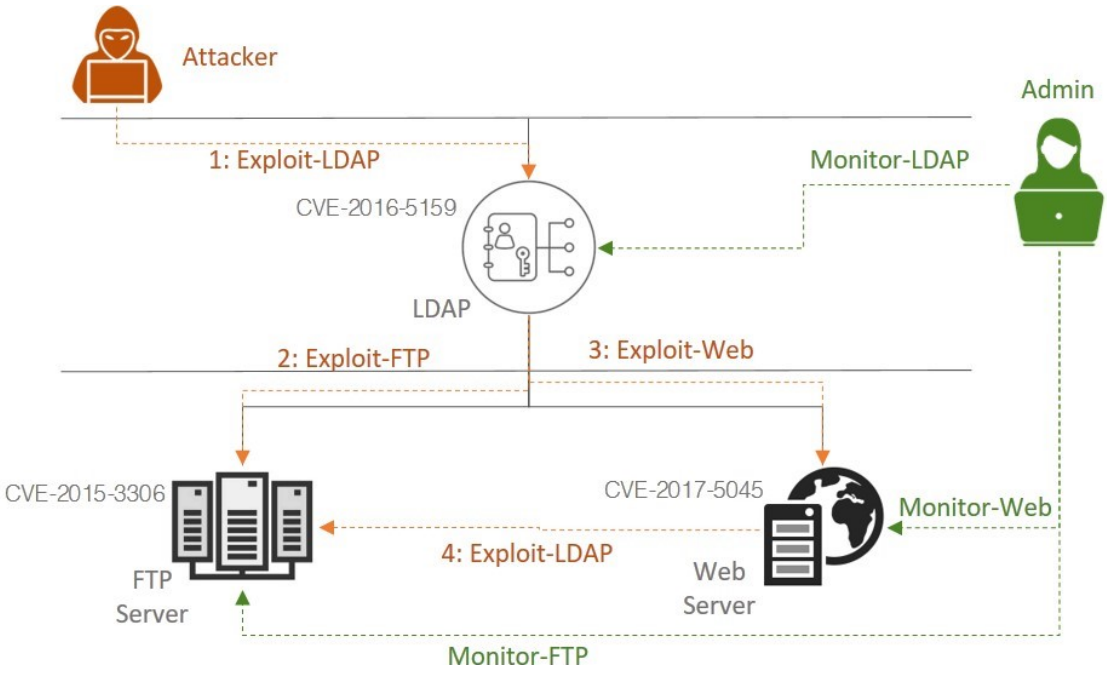


Figure 4.9: An example cloud system highlighting its network structure, the attacker and defender (admin) agents, the possible attacks, and monitoring mechanisms.

cause loss of Confidentiality, Availability, or Integrity (CIA) of that virtual machine (VM). The National Vulnerability Database (NVD) is a public directory of known vulnerabilities and exploits. It assigns each known attack a unique identifier (CVE-id), describes the technology affected, and the attack behavior. Thus, to model the known attacks against our system, we use the Common Vulnerabilities and Exposures (CVEs) listed in NVD.

In the cloud-scenario described in Figure 4.9, we have three VMs– an LDAP server, an FTP server, and a Web server. Each of these servers has a (set of) vulnerability present on it. On the LDAP server, an attacker can use a local privilege escalation to gain root privilege on it. The other two vulnerabilities– A cross-site scripting (XSS) attack on the Web server and the remote code execution on the FTP server– can only

be executed with root access to the LDAP server. We can now describe the threat model for our scenario.

Threat Model

In the example cloud scenario, the attacker starts with user-level access to an LDAP server. The terminal state is to compromise the FTP server (which, as we will see later, leads to an all absorbing state in our Markov Game). The attacker can perform actions such as `1: exploit-LDAP`, `exploit-Web` or `exploit-FTP`. Note that the attacker has two possible paths to reach the goal node, i.e. `priv(attacker, (FTP: root))` which are:

- Path 1: `exploit-LDAP` $\rightarrow$ `exploit-FTP`
- Path 2: `exploit-LDAP` $\rightarrow$ `exploit-Web` $\rightarrow$ `exploit-FTP`

On the other hand, the (network) Admin, who is the defender in our case, can choose to monitor (1) read-write requests made by services running on a VM using host-based Intrusion Detection Systems (IDS) like `auditd`, or (2) network traffic along both the paths using the network-based monitoring agents like `Snort`. We will denote these IDS systems using the terminology `monitor-LDAP`, `monitor-FTP`, etc. We assume that the Admin has a limited budget, i.e., cannot place all possible IDS systems on the cloud network, and thus, must try to perform monitoring in an optimized fashion. On the other hand, the attacker will try to perform attacks along with some path that minimizes their probability of getting detected. Further, we assume an attacker has knowledge of the defender's placement strategy because of the inherent reconnaissance phase in cyber-security scenarios, thus rendering pure strategies for the placement of detection systems useless. Thus, to come up with a good dynamic placement strategy,

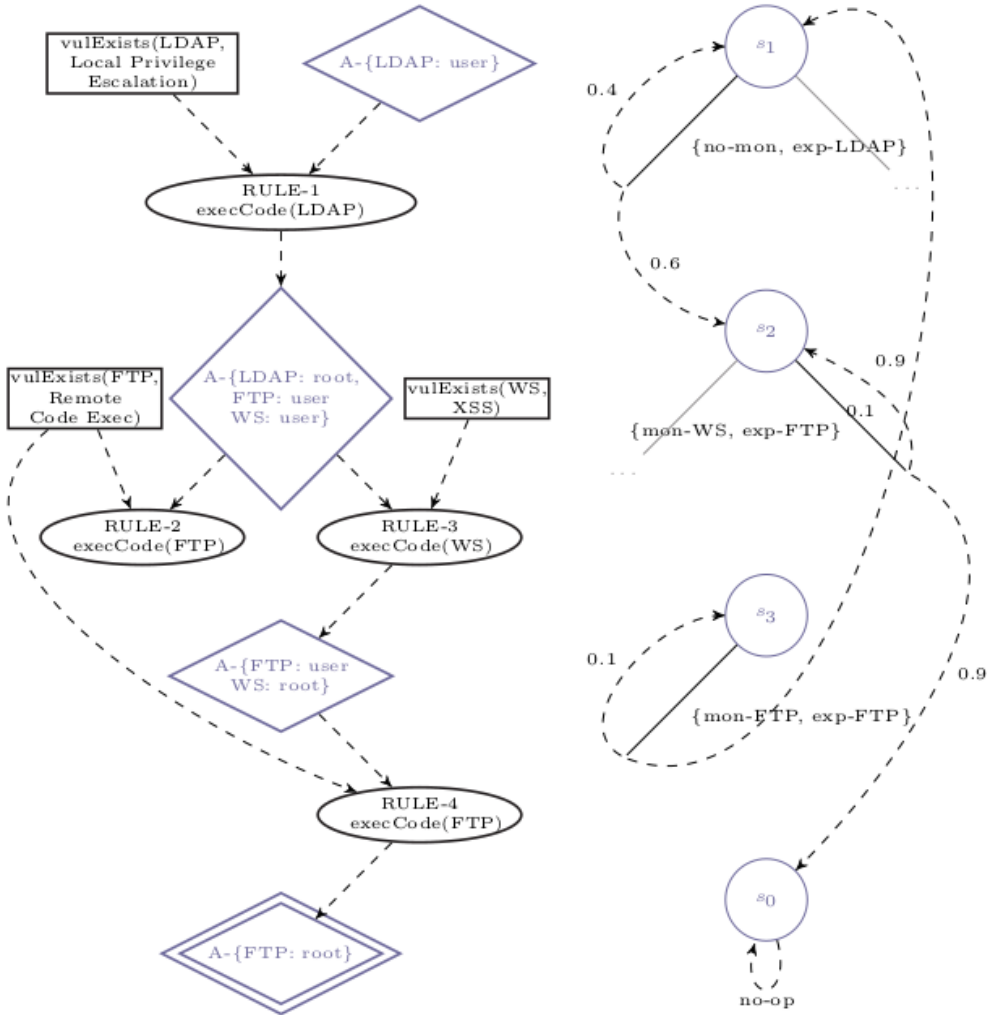


Figure 4.10: The left figure shows the attack graph of the synthetic cloud scenario shown in Figure 4.9. The right figure shows the formulated Markov Game.

we need to model the various multi-attack paths and the attacker's strategy. We first discuss the formalism of Attack Graphs that are a popular way to model the various attacks (and attack paths) in a cloud scenario [94, 24] and then give a brief overview of two-player Markov Games.

Attack Graph Formalism

Attack Graphs (AG) are a representation tool used to model the security scenario of a complex network system like the cloud. Researchers have shown that AG can help to model multi-stage or multi-hop attack behavior.

Attack Graph is a graph $G = \{N, \mathcal{E}\}$ that consists of a set of nodes (N) and a set of edges ($\mathcal{E}$) where,

- As shown in the Figure 4.10, nodes can be of four types– the nodes N_C represent vulnerabilities (shown as rectangles), e.g. `vulExists (LDAP, Local Priv. Escalation)`, N_D represents the attacker's state (shown as diamonds) e.g., `priv(attacker, (LDAP :user))`, rule nodes N_R represent a particular exploit action (shown as ellipses) and finally, root or goal nodes N_G that represent the goal of an attack scenario (shown using two concentric diamonds), e.g., `priv(attacker, (FTP: root))`.

- $\mathcal{E} = \mathcal{E}_{pre} \times \mathcal{E}_{post}$ denotes a set of directed edges in the graph. An edge $e \in \mathcal{E}_{pre}$ goes from a node in N_D or N_C to a rule node N_R and denotes that an attack or rule can only be triggered if all the conditions of the edges going into $n \in N_R$ is satisfied (AND-nodes). An edge $e \in \mathcal{E}_{post}$ goes from a node N_R to $n \in N_D$ indicating the change in attacker's privilege changes upon successful execution of the rule.

Note how the two attack paths mentioned in the threat model section become evident by a simple look at the AG. The conditional and cumulative probability values pertaining to the success of an attack path over the AND (conjunct) and OR (disjunct) nodes can be calculated using probability estimates as described in [109].

Two-Player Markov Games

We now define a two-player Markov Game and also introduce the reader to some notations used later in our modeling. We call the two players of this game the Defender $\mathcal{D}$ (who is the Admin of the cloud system) and the Attacker $\mathcal{A}$ (who is an adversary trying to exploit a vulnerability in the Cloud System). With that, we can now formally define a Markov Game as follows.

Markov Game for two players $\mathcal{D}$ and $\mathcal{A}$ can be defined by the tuple $(S, M, E, \tau, R^{\mathcal{D}}, R^{\mathcal{A}}, \gamma^{\mathcal{D}}, \gamma^{\mathcal{A}})$ [110] where,

- $S = \{s_1, s_2, s_3, \ldots, s_k\}$ are finite states of the game,
- $M = \{m_1, m_2, \ldots, m_n\}$ is the finite set of monitoring actions for $\mathcal{D}$,
- $E = \{e_1, e_2, \ldots, e_{n'}\}$ is the finite set of (exploit) actions available to $\mathcal{A}$,
- $\tau(s, m_i, e_j, s')$ represents the probability of reaching a state $s' \in S$ from the state $s \in S$ if $\mathcal{D}$ chooses to deploy the monitoring m_i and $\mathcal{A}$ chooses to use the exploit e_j,
- $U^i(s, m_i, e_j)$ represents the reward obtained by player $i(= A$ or $D)$ if in state s, $\mathcal{D}$ choose to deploy the monitoring m_i and $\mathcal{A}$ choose to use the exploit e_j,
- $\gamma^i \mapsto [0, 1)$ is the discount factor for player $i(= A$ or $D)$.

In light of recent studies on characterizing attackers based on personality traits [111], one might argue that a defender's perspective of long term rewards is different from that of an attacker. Given that we did not find a formal model or user study clearly stating how these differ, we will consider $\gamma^{\mathcal{A}} = \gamma^{\mathcal{D}} = \gamma$ going forward. As

the solvers for our formulated game can work in cases even when $\gamma^{\mathcal{A}} \neq \gamma^{\mathcal{D}}$, this assumption just helps us simplify the notations.

The concept of an optimal policy in this Markov game is well-defined for zero-sum games [107] where $R^{\mathcal{D}}(s, m_i, e_j) = -R^{\mathcal{A}}(s, m_i, e_j) \ \ \forall \ s \in S, m_i \in M$, and $e_j \in E$. In these cases, a small modification to the Value Iteration algorithm can be used to compute the min-max strategy for both players. To see this, note that the Q-value update for this Markov Game (for player p) becomes as follows,

$$Q^p(s, m_i, e_j) = R^p(s, m_i, e_j) + \gamma \sum_{s'} \tau(s, m_i, e_j, s') \cdot V^p(s') \tag{4.3}$$

where $V^p(s')$ denotes the value function (or reward-to-go) with respect to player p if in state s'. We will use the notation $M(s)$ (and $E(s)$) to denote the set of defender actions (and attacker actions) possible in state s. Given this, the mixed policy x for state s, denoted as $x(s)$, over the defender's applicable actions ($\in M(s)$) can be computed using the value-update,

$$V^{\mathcal{D}}(s) = \max_{x(s)} \min_{e_j} \sum_{m_i} Q^{\mathcal{D}}(s, m_i, e_j) \cdot x(s)_{m_i} \tag{4.4}$$

where $x(s)_{m_i}$ denotes the probability of choosing the monitoring action m_i in state s.

When the Markov Game has a general-sum reward structure and one player can infer the other player's strategy before making their move, the min-max strategy becomes sub-optimal and one must consider other notions of equilibrium [64, 112]. We give an overview of prior work on these lines later in the paper.

Quantifying the Impact of Vulnerabilities

As discussed earlier in this chapter, we will use the Common Vulnerability Scoring System (CVSS) for rating the impact of attacks. For the set of vulnerabilities present in our system, the values of the two metrics are shown in Table 4.4.

VM	Vulnerability	CVE	CIA Impact	Attack Complexity
LDAP	Local Priv Esc	CVE-2016-5195	5.0	MEDIUM
Web Server (WS)	Cross Site Scripting	CVE-2017-5095	7.0	EASY
FTP	Remote Code Exec.	CVE-2015-3306	10.0	MEDIUM

Table 4.4: Vulnerability Information for the Cloud Network.

4.2.2 Markov Game Modeling

Before discussing the game-theoretic formulation in detail, we highlight a few important assumptions. Besides the Markovian assumption, we assume that (1) there is a list of attacks known to both the attacker and the defender (which cannot be immediately fixed either due to lack of resources or restrictions imposed by third-party customers who host their code on the cloud system [14, 35]) and (2) the attacker may reside in the system but will remain undetected until it attempts to exploit an existing vulnerability, i.e. a *stealthy adversary* [34]. These assumptions force our formulation to (1) only deal with known attacks and, (2) come up with good strategies for placing detection mechanisms. The latter is a result of ignoring the partial observability inherent in the problem [51, 52] to come up with scalable solutions, necessary for cloud networks.

States The state of our Markov Game (MG) are derived using the nodes N_D and N_G of an Attack Graph (AG) (the diamond-shaped nodes are mapped to the circular nodes in Figure 4.10). These nodes in the AG represent the state of an attacker in the cloud system, i.e. the attacker's position, and their privilege level. The goal node

N_G of an AG is mapped to a terminal self-absorbing state in our MG while the other nodes N_D represent non-terminal game states. Note that the location of an attacker on multiple physical servers in the cloud network can map to a single state of the AG (and therefore, a single state in our MG). The MG states that map to a goal or a root node N_G of an AG is the terminal self-absorbing states. Among these non-terminal states there exist a set of states S_i that represent the external-facing entry-points to the cloud network, the initial state of any multi-stage attack.

For the cloud scenario shown in Figure 4.9, we have four states. The state s_0 corresponds to the goal node `A-{FTP:root}` and is a terminal state of the MG. The state $s_1 (\subset S_i)$ corresponds to the state where an attack originates while the two states s_2 and s_3 correspond to nodes where the attacker has different access privileges on the three servers (LDAP, WS, and FTP) server. Given that we use a ternary range to denote an adversary's privilege– no-access, user or root-user– on each server, there can be a maximum of nine states (# servers $\times$ # access-levels) [2] in the AG and hence, in our MG. Note that given a set of known attacks, the number of states is often much less (four *vs.* nine for our scenario) because most of the states are not reachable from the states in S_i.

Players and Pure Strategies As mentioned above, the players for our game are the Admin (or the defender) $\mathcal{D}$ and the attacker $\mathcal{A}$. The pure strategy set for $\mathcal{A}$ in state s consists of exploit actions they can perform with the privilege they have in state s. For example, consider s_2 where the he attacker has the access vector `A-{LDAP:root, FTP:user, WS:user}`. With this as the precondition, the attack actions can be represented by the rule nodes N_R (shown in oval) in the AG. Note

[2] Partial observability over the state space can increase the number of states to be a power-set of this number, i.e. $2^{(\#\text{ servers} \times \#\text{ access-levels})}$

that there is always a vulnerability node $\in N_C$ associated with a rule node and thus, with each action.

The pure strategy set for $\mathcal{D}$ in a state s consists of monitoring actions where each such action corresponds to placing an Intrusion Detection System (IDS) for detecting attacks that can be executed by $\mathcal{A}$ in state s. These actions are made possible in real-world scenarios by using two sorts of IDS systems– (1) host-based IDS like *auditd* that can notify $\mathcal{D}$ about an abnormality in the use of CPU resources or access to files on the server and (2) network-based IDS like *snort* that can observe traffic on the wire and report the use of unexpected bit patterns in the header or body of a packet. Although a pure strategy for $\mathcal{D}$ can only detect a single attack in our simple example, it is possible that a set of detection systems, capable of detecting multiple attacks, is considered a pure strategy. We will see this in the case of the real-world cloud scenario discussed in the experimental section. In the context of Stackelberg Security Games, such groups of actions are often called schedules [102, 103], and the pure strategy is defined over these schedules. We note that our modeling supports such representations. [3]

To allow for a realistic setting, we add one more action to the pure strategy set of each player–`no-act` and `no-mon`. These represent the no-op for each player which allows an attacker to not attack in a particular state if it feels that there is a high risk of getting caught. Similarly, this allows a defender to not monitor for an attack thereby saving valuable resources.

[3]In these cases, the Subset of Sets are Sets (SSAS) property defined in [108] may not hold and thus, the Strong Stackelberg Equilibrium will not always be a Nash Equilibrium for the formulated Markov Game (see later (*Lemma 1*) for details).

Transitions The transitions in our MG represent that given a particular state and a pair of actions drawn from the joint action space $E \times M$, the probability with which a game reaches a state s', i.e. $\tau(s, m, e, s')$. There exists a few obvious constraints in the context of our MG– (1) $\tau(s, m, \texttt{no-act}, s) = 1$, i.e. if an attacker does not execute an attack, the game remains in the same state, (2) when $e \neq \texttt{no-act}$, $\tau(s, m_e, e, s') = p/|S_i| \ \forall s' \in S_i$ where p is the probability that e is detected by m_e, the monitoring service deployed to detect e, i.e. when successfully detected, the attacker starts from either of the initial states with equal probability, and (3) $\tau(s, \texttt{no-mon}, e, s') \neq 0$ if $s \notin S_i$ and $s' \in S_i$, i.e. an attacker cannot be detected if the defender does not perform a monitoring action.

We highlight a few transitions for our Markov Game in Figure 4.10. In state s_1, the defender does not monitor the exp-LDAP attack action and with 0.6 probability the game moves to the state s_2 (and with 0.4 it remains in s_1). These probability are calculated using the Access Complexity vector and the function defined in [109] for obtaining the probability of success given an attack. This is also done when the defender deploys a monitoring mechanism m_e but the attacker executes another attack e' where $e \neq e'$ and $e' \neq \texttt{no-act}$ (see the transition for an example joint action from s_2 in Fig 4.10). Lastly, the transition from s_3 shows a case relating to (3) mentioned in the previous paragraph. The value of $\tau(s_3, \text{mon-FTP}, \text{exp-FTP}, s_1) = 0.9$ because monitoring access to files like `/etc/passwd` and `/etc/shadow` can only detect some forms of privilege escalations (such as remote code execution that tries to eithers creates a new user or tries to escalate the privilege of a non-root user), but may not be able to monitor a case where an attacker simply obtains access to a root user account.

		$\mathcal{D}$ (Defender)		
		no-mon	mon-Web	mon-FTP
	no-act	$0, 0$	$0, -2$	$0, -3$
$\mathcal{A}$ (Attacker)	exp-Web	$7, -7$	$-8, 6$	$7, -10$
	exp-FTP	$10, -10$	$10, -12$	$-8, 5$

Table 4.5: Utilities $(R^{\mathcal{A}}, R^{\mathcal{D}})$ for the state s_2.

	no-mon	mon-LDAP
no-act	$0, 0$	$0, -2$
exp-LDAP	$5, -5$	$-5, 3$

	no-mon	mon-FTP
no-act	$0, 0$	$0, -2$
exp-FTP	$10, -10$	$-8, 6$

Table 4.6: Utilities $(R^{\mathcal{A}}, R^{\mathcal{D}})$ for states s_1 (left) and s_3 (right).

Rewards The rewards that can be obtained by the players depending on their strategy in a particular state (except the terminal state s_0) of our cloud scenario is shown in Table 4.6 and Table 4.5. Most prior works [13, 52, 34] have used (weak) heuristics to come up with attacker utility and or defender's resource costs. In our case, the reward values are obtained using multiple metrics– (1) the impact score (IS) of a particular attack, (2) the cost of performance degradation (provided by security and engineering experts in the cloud domain; often obtained by running MiniNET simulation) based on the placement of a particular IDS [35], and (3) the number of hops taken by an attacker to reach a particular state, which is often used to measure how advanced an Advanced Persistent Threat (APT) is. Note that the last factor is the non-Markovian part of the overall reward function of our Markov Game; it depends on the path taken by an attacker to reach a particular state. To bypass this issue, we consider all possible paths an attacker can take to reach the particular state and average the path value, which gives us an average of how advanced an APT is.

Further, the actual path taken by a stealthy adversary who has been residing in the network for a long time is difficult (if not impossible) to obtain. Thus, we believe an average estimate provides a good heuristic for estimating the importance of an APT.

We now explain how the reward value for the action pair `(mon-Web, exp-Web)`, shown in Table 4.5, was obtained. First, The impact score for this vulnerability CVE-2017-5059, shown in Table 4.4, is 7. Second, we monitored performance using `Nagios` [113] to measure the end-to-end network bandwidth, the number of concurrent requests to live web-services, and the delay in servicing network requests when mon-Web was deployed. We observed that there was an increase in network delay, a decrease in network bandwidth, and a decrease in the number of concurrent requests serviced per second. Based on expert knowledge, we estimated the reward (or rather impact on performance) of placing the IDS that monitors this vulnerability is -2. Finally, given that this vulnerability can only be executed if the attacker has exploited at least one vulnerability before coming to this state, the APT score was calculated to be 1. Thus, the defender's reward for placing the correct IDS that can detect the corresponding attacker action is 7 minus 2 (cost incurred due to reduced performance) plus 1 (for detecting an APT that had already penetrated 1-hop into the network), totaling 6. On the other hand, the attacker's reward for this action pair is -7 spending effort is executing a vulnerability of impact 7 plus -1 for losing a vantage point in the cloud network, totaling a reward of -8. The other reward values were defined using a similar line of reasoning. Given that the defender's cost of placing IDS is not of any concern for the attacker [4] , when an attacker chooses `no-act`, $\mathcal{A}$'s reward is 0. On the contrary, the defender will still incur a negative reward if it deploys a monitoring system because it impacts the performance of the sub-net.

[4]This is a strong reason to move away from the zero-sum reward modeling in [38].

Algorithm 2 A Dynamic Programming Approach to find the Strong Stackelberg Equilibrium in Markov Games

1: **procedure** GIVEN $(S, M, E, \tau, R^{\mathcal{D}}, R^{\mathcal{A}}, \gamma^{\mathcal{D}} = \gamma^{\mathcal{A}} = \gamma)$,

2: OUTPUT$(V^i(s), \pi^i(s)\ \forall\ i \in \{\mathcal{A}, \mathcal{D}\})$

3: $V(s) = 0\ \forall\ s$

4: **while** *count* $< k$ **do**

5: // Update Q-values

6: Update $Q^{\mathcal{D}}(s, m, e)$ and $Q^{\mathcal{A}}(s, m, e)\ \ \forall\ s \in S, m \in M(s), e \in E(s)$

7: using $R^{\mathcal{D}}$, $R^{\mathcal{A}}$ and $V(s)$.

8: // Do value and policy computation

9: Calculate $V^i(s)$ and $\pi^i(s)$ for $i \in \{\mathcal{A}, \mathcal{D}\}$ using the values $Q^i(s, m, e)$ in Equation 4.5

10: *count* $\leftarrow$ *count* $+ 1$

11: **end while**

12: **end procedure**

Optimal Placement Strategy

Finding the optimal solution to a two-player general-sum Markov Games is more involved than finding the min-max strategy (in zero-sum settings). Moreover, in our threat model, we assume that the attacker $\mathcal{A}$, with reconnaissance efforts, will get to know the strategy of the defender $\mathcal{D}$, imparting it the flavor of leader-follower games.

We highlight a dynamic programming approach shown in Algorithm 2. Although this algorithm looks similar to the one used for computing min-max equilibrium, it has an important difference. In line 9, instead of using Equation 4.4 to calculate the optimal value and player strategies, we compute the Strong Stackelberg Equilibrium (SSE) in each state. For each iteration, we first consider the Q-values for that state

and the joint actions represented as a normal-form game matrix. We then find the optimal policy for both players in state s. Since our model, at present, does not consider multiple adversary types, the equilibrium calculation for each state can be done in polynomial time [102]. This type of a solution resembles the idea of finding Stackelberg Equilibrium in discounted stochastic games, which has been discussed earlier in [64]. As the authors compile the solution into an ILP approach over all the states and time horizon of the Markov Game, it becomes computationally intensive in our case given a large number of states in our formulation. Moreover, our dynamic programming approach acts as an anytime solution that can be stopped at a premature stage (by setting lower values of k in Algorithm 2) to yield a strategy for placement.

Given that the defender has a mixed strategy in every state, the notation x that denoted their policy in the previous chapters is indexed with s and denoted as $x(s)$. Similarly, the attacker's policy is denoted as $q(s)$. We now highlight the MIQP used in [61] that we use for the value function update in Algorithm 2.

$$\max_{x(s),q(s)} \sum_{m\in M}\sum_{e\in E} Q^{\mathcal{D}}(s,m,e)x(s)_m q(s)_e \tag{4.5}$$

$$\begin{aligned}
s.t. \quad \sum_{m\in M} x(s)_m &= 1,\ \forall x(s)_m,\ x(s)_m \in [0,1] && \text{Defender's selects a valid mixed strategy.}\\
\sum_{e\in E} q(s)_e &= 1,\ \forall q(s)_e,\ q(s)_e \in \{0,1\} && \text{Attacker's selects a valid pure strategy.}\\
0 \le v - \sum_{m\in M} Q^{\mathcal{A}}(s,m,e)x(s)_m &\le (1-q(s)_e)L\ \ \forall q(s)_e && \text{Attacker's pure strategy maximizes their reward given defender's mixed strategy.}
\end{aligned}$$

where L is a large positive number.

The assumption that an attacker, with the inherent advantage of reconnaissance, is aware of the defender's mixed policy in each state can be a strong one in the context

of Markov games. Thus, one might question the optimality of the strategy that we come up with using Algorithm 2. In [114], researchers have shown that SSEs are a subset of Nash Equilibrium for a particular class of problems. Specifically, they show that if the security resource allocation problem (which in our case, is allocating IDS for covering a vulnerability) has a particular property, termed as SSAS, then the defender's SSE is also a NE for the game. Given this, we can state the following. [5]

Lemma 4.2.1. *If the Subset of Set Are Sets (SSAS) property holds in every state s of a Markov Game, then the SSE is also a NE of the Markov Game.*

Proof. We will prove the lemma by contradiction. Let us assume that SSAS property holds in every state of a Markov Game (MG), but the SSE of the MG is not the NE. First, consider $\gamma = 0$ for this MG. Thus, the SSE and NE strategy for each state can be calculated only based on the utilities of only this state. Now, if SSE $\notin$ NE for this Markov Game, then there is some state in which the SSE strategy is not the NE strategy. But if that is the case, then we would have violated the SSAS theorem in [114] for the state, which cannot be true. For the case $\gamma > 0$, the proof still holds because note that the SSAS property is not related to the Q-values using which the strategy is computed in the Markov Game. □

Note that this holds trivially for our small example because the defender's pure strategy for each state is to deploy a single IDS (and thus, all subset of schedules are are a possible schedule).

[5] In the case of multiple attackers, SSE $\notin$ NE. Although such scenarios exist in cybersecurity settings, we consider a single attacker in this modeling and plan to consider the multiple attacker setting in the future.

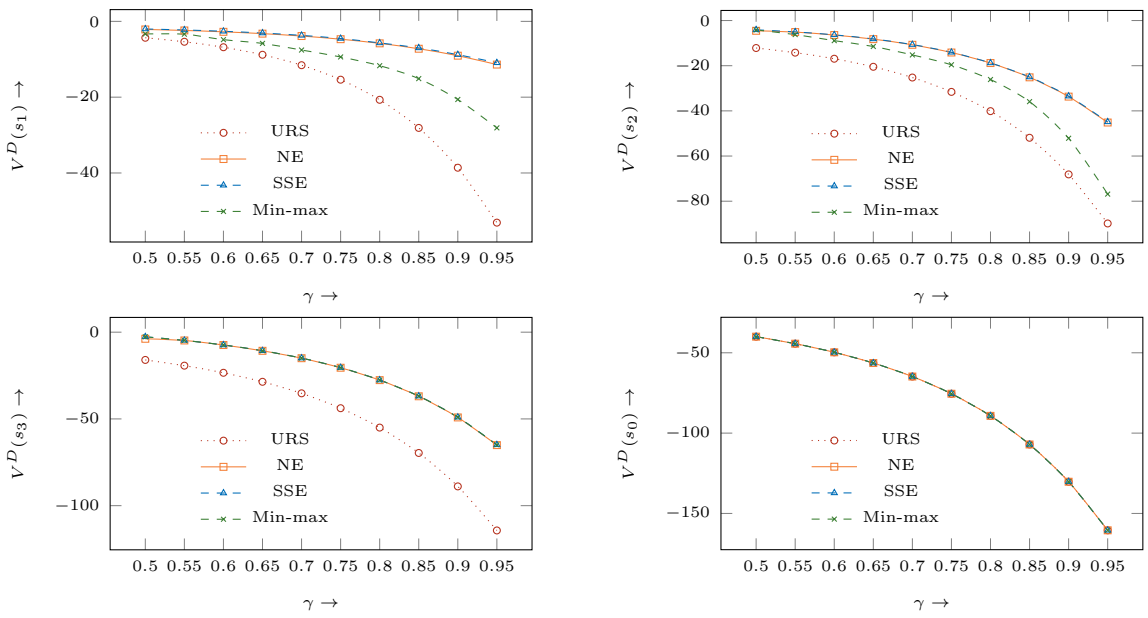


Figure 4.11: Defender's value for each of the four state–s_1 (top-left), s_2 (top-right), s_3 (bottom-left), and s_0, which in the all absorbing terminal state (bottom-right).

4.2.3 Experimental Results

In this section, we first compare the effectiveness of the optimal strategies for our general-sum leader-follower Markov-game against the Uniform Random Strategy, which is a popular method and often used as a baseline for Moving Target Defense strategies in cybersecurity. We then discuss the set-up of a real-world, small scale cloud scenario and the gains we can obtain using our formulated Markov Game.

Evaluation of Strategies

We first discuss two baseline strategies and then briefly explain why we choose it for comparison with the SSE strategy for our general-sum leader-follower Markov-game formulated in the previous section.

• *Uniform Random Strategy (URS)* In this, the defender samples an action by drawing from a uniform random distribution over pure-strategies. For example, in the state s_1 shown in Table 4.5, the defender can choose to monitor the FTP or the Web server or neither of them, all with an equal probability of 0.33.

Researchers have claimed that selecting between what to choose when shifting between MTD configurations should be done using a uniform random strategy [1]. Although there have been other methods based on Stackelberg Security Games (SSGs) which have shown that such a strategy may be sub-optimal [35], it provides a baseline strategy that can be used in the context of our Markov Game. Adapting the latter strategies proposed in previous work needs us to compile our multi-stage Markov into a single step normal form game. First, there is no trivial way of doing this conversion as the rewards in [35] consider the average impact on the network which is difficult to encode meaningfully in our Markov Game. Furthermore, the pure strategy set in [35] would have to incorporate full attack paths as opposed to single attack actions. This would make the strategy computation time-consuming. Second, our work can be seen as a generalization of applying the notion of Stackelberg Equilibria, similar to [64], for Markov Games in the context of IDS placement and thus, a counterpart solution to the normal form games case described in [35] to Markov Games. Hence, we do not consider [35] as a baseline.

• *Min-max Strategy* Although our game is a general sum setting, one might ask how sub-optimal the min-max strategies for a similar zero-sum Markov game is when we ignore the impact on performance. In essence, the attacker still has the same utility in the individual states shows in Tables 4.6 and 4.5, but the defender's reward values are just the opposite of the attacker's reward, making it a zero-sum game. Here,

we hope to see that the impact on performance would reduce the defender's overall utility.

Comparison of Strategies In Figure 4.11, we plot the values of the four states ($V(s)$) of our game for the baselines (URS and Min-max), and the Strong Stackelberg Eq. (SSE). We also, to provide empirical support for Lemma 1, plot the Nash Eq (NE). On the x-axis, we vary the discount factor and on the y-axis plot the value of the state with respect to the defender. In the terminal state s_0, the defender gains a high negative reward because the attacker was able to exploit all the possible vulnerabilities successfully without getting detected. Thus, for all the states, since there is a non-zero probability of reaching s_0, the defender's value function is negative. Note that as one weighs the future rewards more, i.e. γ approaches 1, the value of the states decreases in magnitude because the negative reward in s_0 is given higher weight.

As stated above, the SSE for our example game is the same as the NE for our Markov Game, and the curves for both the strategies overlap. On the other hand, URS is much worse off than our strategy for all the states with more than a single action whereas, Min-max, although better than URS, is sub-optimal with respect to SSE for all states except s_3 and s_0. s_0, being a terminal state, has only one action for the defender and thus, all the methods are trivially equivalent. Thus, all the curves overlap (bottom-right in Figure 4.11). In state s_3, which is just a single action away from the terminal state with high negative reward, the defender always picks the action to monitor for an attack regardless of the performance impact (whose magnitude is less in comparison to the impact of the attack). Thus, even though the Min-max strategy is ignorant of the latter costs, it picks the same strategy as SSE. Hence, their plots overlap (bottom-left in Figure 4.11). Now, before discussing the

differences between SSE and Min-max in the other two states (s_1 and s_2), we take a look at the mixed strategy obtained by finding the SSE of our game. This will help us in explaining the sub-optimality of the Min-max strategy. For a discount factor of $\gamma = 0.8$:

```
1 π_MG-SSE(s_0) : {terminate: 1.0}
2 π_MG-SSE(s_1) : {no-mon: 0.097, mon-LDAP: 0.903}
3 π_MG-SSE(s_2) : {no-mon: 0.0, mon-Web: 0.539, mon-FTP: 0.461}
4 π_MG-SSE(s_3) : {pi_no-mon: 0.0, mon-FTP: 1.0}
```

Note that, in our example, barring the terminal state s_0, other states have only one or two proper detection actions because `no-mon` asks the defender to not monitor for any attacks. Thus, we expected that these actions will have probabilities almost equal to zero (unless it has a considerable impact on performance). In the case of state s_1, the 0.097 probability of picking that action shows that in states far away from the terminal, the defender chooses to be a little more relaxed in terms of security and pays more attention to performance. On the contrary, in state s_3 an exploit action will move the game to the terminal state that has a high-negative reward for the defender, and thus, the defender is asked to place all attention to security. In general, this implies that an Admin $\mathcal{D}$ should pay more attention to security against APTs deep within the network in states closer to a critical resource and can choose to reason about performance near the entry points of the cloud system. Thus, in these states, the optimal mix-max strategy, oblivious to the performance costs, invests in monitoring resources and thus, becomes sub-optimal with respect to the SSE.

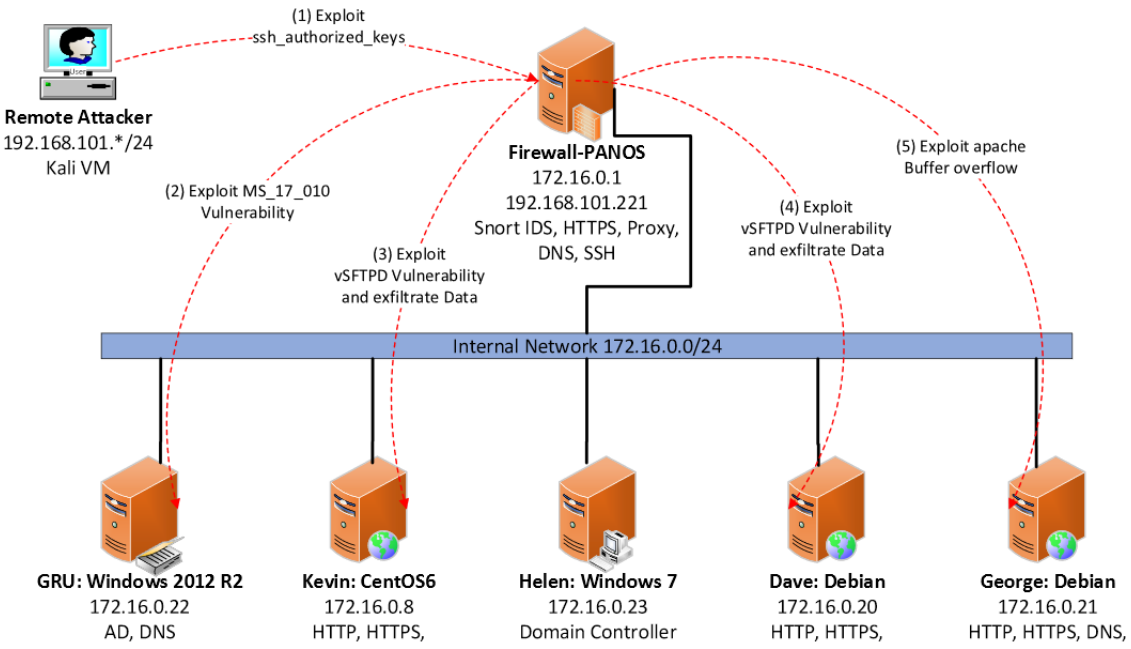


Figure 4.12: A small-scale cloud system that emulates the setup used in the Western Region Cyber-security Defense Competition.

Case Study on the Cloud

In this section, we do a case study on a real-world sub-network in a cloud system, highlighting briefly the system setup, the Markov Game formulation, the comparison between URS and SSE for a (cherry-picked) state and how all these strategies can be implemented with the help of Software Defined Networking.

Implementation details. We utilized the Virtual Machine (VM) images from the Western Region Cybersecurity Defense Competition (WRCCDC) [115]. The competition helps university students (the Blue Team) gain the first-hand experience in dealing with real-world attacks. Students are asked to defend a corporate infrastructure against experienced white-hat hackers (the Red-Team). In the scenario shown in Figure 4.12, a Red Team attacker mounts a multi-step attack by following a slow and low approach, that tries to evade the IDS placed.

Host	High	Medium	Low
172.16.0.22	4	14	1
172.16.0.23	2	3	1
172.16.0.8	3	8	3
172.16.0.16	0	13	6
172.16.0.20	0	2	1
172.16.0.11	0	1	2
172.16.0.1	0	0	1
172.16.0.21	0	0	1
Total	**9**	**41**	**16**

Table 4.7: The number of vulnerabilities found in each machine of our cloud system via persistent exploration with OpenVAS vulnerability scanner.

The goal of the attacker can be either to disrupt the network services or ex-filtrate data out of the private networks. Both of these attacks, if successful, can lead to the loss of mission-critical information (FTP server files are valuable to the company) and business (service downtime). We model each of these goal states in the attack graph as states that lead to an all-absorbing terminal state with unit probability, thus ending the game.

The set of attacks were discovered using low-intensity network scanning tools like OpenVAS that run over an extended period of time and generate a report of the vulnerabilities present in the system. Due to space considerations, we summarize the vulnerabilities present in our cloud network in Table 4.7. Corresponding to the attacks, we considered the deployment of IDS mechanisms like Snort IDS, Web Proxy, etc. situated at different levels of the protocol stack. We used WRCCDC's VM

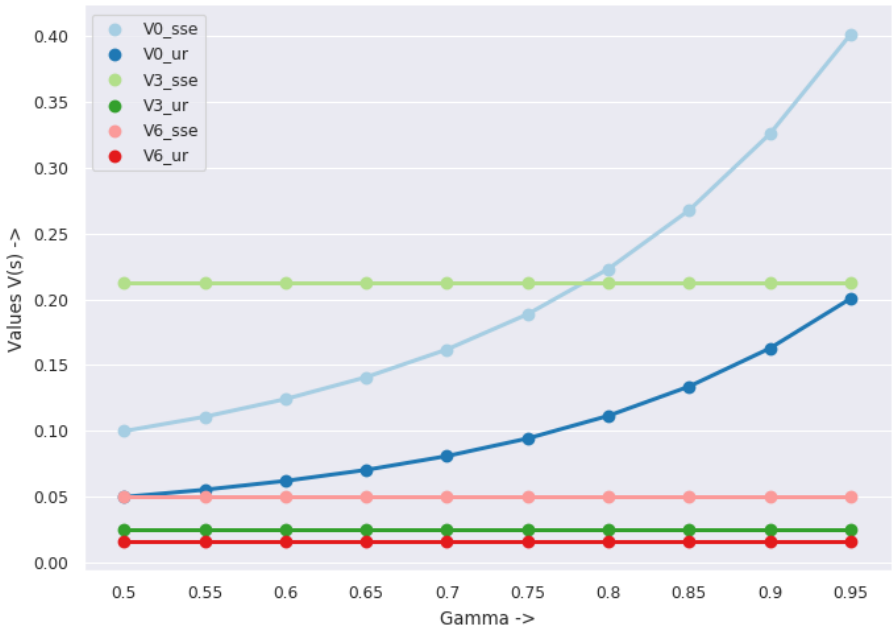


Figure 4.13: Defender's value in the states s_0, s_3 and s_6 depending on the strategy they follow as the discount factor γ increases from 0.5 to 1.

images to create a similar environment in our organization's cloud service. To connect these VMs, we created a flat structure network with *Palo-Alto Network OS (Next-Generation Firewall)* hosted at the gateway of the network (172.16.0.0/24) and had eight host machines in total [116].

The network was connected using SDN switch with OpenFlow v1.3 for (1) vulnerability scanning to gather knowledge about known attacks in the cloud, (2) computing the Markov Game strategy and (3) enforcing a particular deployment strategy and switching to a new one after a fixed time period T. For the first step, we used scanning tools like OpenVAS, as described earlier. For the second step, we use our strategy computation algorithm that also solves the optimization problem using Gurobi solver. For the last step, we used to enable and disable (kill) scripts for the different IDS mechanisms and perform them using SDN protocols.

Results In the formulated Markov Game, we have eight states corresponding to each VM in our cloud system. In figure 4.13, we highlight the defender's value for states s_0, s_3 and s_6. In state s_0, the attacker has root privilege on the Firewall VM which is a lucrative vantage point in the network as it can help the adversary reach all other states by choosing from an array of vulnerabilities. The defender's pure strategies in this state correspond to deploying a set of IDS mechanisms (as opposed to a single IDS); certain sets are profitable in terms of resource usage and effort needed for configuring them. For example, deployment of two Network-based IDS using Snort is easier to configure for the Admin than the deployment of a host-based and a network-based IDS at the same time. In the other states such as s_3 and s_6, the attacker can only leverage exploits to gain privileges in the systems associated with those stages. Thus, successful exploits lead to a terminal stage.

For this study, we did consider a zero negative utility in the terminal state. Thus, the defender's value in all the states is positive. Further, the rewards obtained in all other states except s_1 remain constant regardless of the value of γ. If Figure 4.13, the values in s_3 and s_6, which are solely based on the impact of the exploit and performance considerations, are always higher when using the inferred strategies at Strong Stackelberg Equilibrium (SSE) when compared to Uniform Random Strategy. Similarly, in s_1, our proposed strategies outperform URS but owing to the various states that can be reached from this vantage point, the values increase with an increase in γ. URS neither discriminates between attacks that have higher impacts nor cares about the asymmetric performance impacts; this becomes a major reason for its sub-optimality.

4.3 Related Work

Similar to our work in Chapter 3, Moving Target Defenses that shift the attack-surface in the context of cloud networks has been extensively investigated [117, 34, 48, 109]. While these methods seek to take away the advantage of an attacker's reconnaissance, they can prove to be less effective against a stealthy adversary, i.e. one who can reside deep within the network [99]. An overview of such works can be found in [7]. While our methods in this chapter consider the movement of the detection surface, the proposed game-theoretic modeling can be generalized for the movement of the attack surface to address the challenges posed by a stealthy adversary.

In the context of designing meta-defenses for the detection surface, [94] has considered finding the optimal placement of detection systems in large networks. Unfortunately, they consider a static placement that becomes insecure as the adversary gathers more knowledge about their static (or pure) placement strategy. On the other hand, authors in [34] consider a dynamic (or mixed) placement strategy based on graph-theoretic measures obtained by modeling the large network as a graph. As this method incorporates neither the knowledge of known vulnerabilities nor models an adversary, it solely optimizes for performance. In this regard, our work in this chapter seeks to find movement strategies that strike a balance between the security and performance metrics when considering the placement of detection systems in the cloud.

As shown in Chapter 3, designing a movement strategy M needs to incorporate attacker modeling and consider game-theoretic reasoning for it to be effective. While an array of work for physical security environments considers the concept of Stackelberg equilibrium, called Stackelberg Security Games [118, 119, 120, 121], researchers have also looked into similar scenarios for placement of honey-nets in cloud-network settings [14, 4]. Unfortunately, the latter works that also find the Stackelberg equilib-

rium, do not scale when the number of defender strategies explodes combinatorially. Fortunately, researchers have shown that decomposition of the reward structure makes the problem of finding the Stackelberg Equilibrium computationally efficient [101]. In our work in Section 4.1, we leverage this information to design the rewards for our game and ensure that the Stackelberg equilibrium balances between two important metrics [98]– (1) the costs of placing IDSs (on performance, cost of countermeasure deployment, etc.) and (2) the impacts on the security of our system.

In the context of our work in Section 4.2, researchers have relied on the use of attack graphs in trying to come up with a defender's strategy of modifying the design of the cloud environment [122, 123, 124]. While these make use of automated planning and MTD formalism, they cannot be easily adapted for dynamic defenses like MTD. In the context of large-scale cloud systems, the generation of attack graph suffers from scalability issues [94]. To solve this problem in Section 4.2, we use a polynomial-time distributed attack graph generation method developed in [24]. In comparison with [64], who formulate problems in physical security settings as a Stochastic Game and show that Markov strategies may be arbitrarily sub-optimal, our scenario can be modeled as a Markovian setting. Further, our proposed approach based on dynamic programming is more scalable and has flavors of an anytime search method.

4.4 Concluding Remarks

In Section 4.1, we addressed the problem of placing a fixed number of IDS systems in a large cloud environment by proposing a Moving Target Defense (MTD) approach for shifting the detection surface. While we also designed a similar approach for moving the attack surface in web-applications in Chapter 3; this work was different in two regards– (1) The main objective in this setting was to optimize performance while

in the earlier work, the primary concern was security and (2) the computation of the Stackelberg equilibrium, even without uncertainty w.r.t. attackers, became difficult with a general reward structure because the size of MTD configurations in C exploded combinatorially. Thus, we have to resort to a problem-specific reward description that allowed us to effectively design movement strategies M in this context.

In Section 4.2, we considered a more pragmatic threat model in the cloud in which attackers can plan multi-stage attacks. In this case, we leveraged the attack-graph of a cloud network to design a Markov Game between the defender and the attacker. The optimal defender policy provided a trade-off between system performance and security of critical resources on the network. As an added advantage, the state space of the Markov Game divided the network system into disjoint parts; this helped us address the problem of asymmetric performance impact in the cloud network.

Chapter 5

LEARNING MOVEMENT STRATEGIES FOR MOVING TARGET DEFENSE IN WEB-APPLICATION AND CLOUD NETWORK SECURITY

C	(Any) Surface Shifting
t	Constant Time Period
M	Learns the Strong Stackelberg Strategy in Bayesian Stackelberg Markov Games

---

In the previous chapters, we model the interaction between a defender and a rational adversary in Moving Target Defenses for web-application and cloud-network security as a game. This helps us leverage resources curated by system administrators and cyber-security experts and define the various parameters of the game. We can then design various optimization problems that can infer the movement strategies at Strong Stackelberg Equilibrium in these games. Unfortunately, expecting experts to provide detailed models about the system dynamics and the rewards may often be difficult, if not unrealistic. At the same time, one may have access to a simulator for the environment where they can execute defense and attack actions and infer its effects. In such scenarios, we may be able to learn a movement policy from scratch or adapt an inferred policy.

In this chapter, we first propose a unifying game-theoretic framework termed as the Bayesian Stackelberg Markov Games (BSMGs). BSMGs can model all the nuances of the MTDs described in the previous chapters– it can (1) capture the uncertainty over attacker types, (2) handle multi-step attacks, and (3) model switching defenses. We then define the notion of optimal strategy as the Strong Stackelberg Equilibrium of BSMGs and consider how they can be learned.

Researchers in security have considered techniques in reinforcement learning to learn optimal movement policies over time [2, 54, 55, 56]. Unfortunately, these works ignore (1) the strategic nature (and the rational behavior) inherent in an adversary setting, and (2) the incomplete knowledge a defender may possess about their opponent. This, as we will show in our experiments, results in a new attack surface where the defender's movement policy can be exploited by an adversary. To mitigate this, we bridge the knowledge gap between existing work, and techniques in multi-agent reinforcement learning by proposing a Bayesian Strong Stackelberg Q-learning (BSS-Q) approach. First, we can show that BSS-Q converges to the Strong Stackelberg Equilibrium of BSMGs. Second, we design an Open-AI gym [125] style multi-agent environment for two Moving Target Defenses (one for web-application and the other for cloud-network security) and compare the effectiveness of policies learned by BSS-Q against existing state-of-the-art static policies and other reinforcement learning agents.

In the next section, we motivate the need for a unifying framework and formally describe the proposed game-theoretic model of BSMGs. We briefly discuss how the two Moving Target Defenses defined in Chapter 3 and Chapter 4 can be modeled as BSGMs. We then introduce the Bayesian Strong Stackelberg Q-learning approach and show that it converges to the SSE of BSMGs, followed by a section showcasing experimental results. Finally, before concluding, we discuss related work.

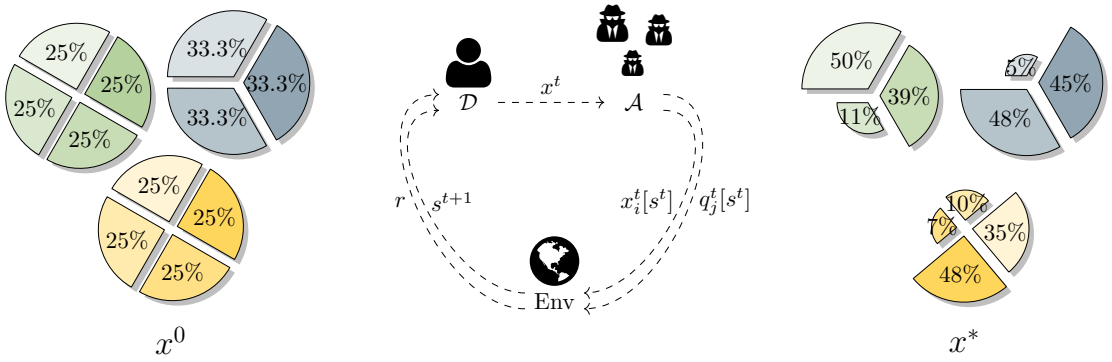


Figure 5.1: The defender starts with a uniform random strategy (x^0) and then interacts with the environment via a Bayesian Strong Stackelberg Q-learning (BSS-Q) approach that modifies the defender's movement strategy at every step based. A simulated and rational adversary in this Stackelberg setting, is aware of the defender's commitment (i.e. the defender's mixed strategy). After numerous episodes, the learned movement policy converges to the Strong Stackelberg Eq. (SSE) of the BSMG yielding x^*.

5.1 Bayesian Stackelberg Markov Games (BSMGs)

Markov Games (MGs) [110], defined in the previous chapter, are used to model multi-agent interactions in sequential planning problems. Under this framework, a player can reason about the behavior of other agents (co-operative or adversarial) and come up with policies that adhere to some notion of equilibrium (where no agent can gain by deviating away from the action or strategy profile). While MGs have been widely used to model adversarial scenarios, they suffer from two major shortcomings– (1) they do not consider incomplete information about the adversary [126, 64, 42, 2] and/or (2) they consider weak threat models where the attacker has no information about the defender's policy [53, 54]. On the other hand, Bayesian

Stackelberg Games [61, 20] is a single-stage game-theoretic formalism that addresses both of these concerns but cannot be trivially generalized to sequential settings.

To overcome these challenges of expressiveness required for Moving Target Defenses (MTDs) while ensuring scalability, we introduce the formalism of Bayesian Stackelberg Markov Games (BSMGs). BSMGs extends Bayesian Stackelberg Games (BSGs) to multi-stage sequential games. While one can consider using existing formalism in Markov Games that capture incomplete information, they face severe scalability issues and have thus been unpopular in cyber-security domains (we discuss how BSMG is situated in this landscape of works in Section 5.4). In the context of Moving Target Defense (MTD), BSMG acts as a unifying framework helping us characterize optimal movement policies against strategic adversaries, capture transition dynamics and costs of the underlying cyber-system, aid in reasoning about stronger threat models, and consider incomplete information about strategic adversaries. Formally, a BSMG can be represented by the tuple $(P, S, \Theta, A, \tau, U, \gamma^{\mathcal{D}}, \gamma^{\mathcal{A}})$ where,

- $P = \{\mathcal{D}, \mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2, \ldots \mathcal{A}_t\}\}$ where $\mathcal{D}$ denotes the leader (defender) and $\mathcal{A}$ denotes the follower (attacker). In our model, only the second player has t types.
- $S = \{s_1, s_2, \ldots, s_k\}$ are k (finite) states of the game,
- $\Theta = \{\theta_1, \theta_2, \ldots \theta_k\}$ denotes k probability distributions (for k states) over the t attackers and $\theta_i(s)$ denotes the probability of i-th attacker type in state s
- $A = \{A^{\mathcal{D}}, A^{\mathcal{A}_1}, \ldots A^{\mathcal{A}_t}\}$ denotes the action set of the player and $A^i(s)$ represents the set of actions/pure strategies available to player i in state s.
- $\tau^i(s, a^{\mathcal{D}}, a^{\mathcal{A}_i}, s')$ represents the probability of reaching a state $s' \in S$ from the state $s \in S$ when the $\mathcal{D}$ chooses $a^{\mathcal{D}}$ and attacker type i choose the action $a^{\mathcal{A}_i}$,

- $U = \{U^{\mathcal{D}}, U^{\mathcal{A}_1}, \ldots, U^{\mathcal{A}_t}\}$ where $U^{\mathcal{D}}(s, a^{\mathcal{D}}, a^{\mathcal{A}_i})$ and $U^i(s, a^{\mathcal{D}}, a^{\mathcal{A}_i})$ represents the reward/utility of $\mathcal{D}$ and an attacker type $\mathcal{A}_i$ respectively if, in state s, actions $a^{\mathcal{D}}$ and $a^{\mathcal{A}_i}$ are chosen by the players,

- $\gamma^i \mapsto [0, 1)$ is the discount factor for player i. We will assume that $\gamma^{\mathcal{D}} = \gamma^{\mathcal{A}_i} = \gamma$.

In BSMGs the individual stage games constitute normal-form Bayesian games with a distribution over attacker types; this is in contrast to the unit probability over a single adversary type in MGs. Both in physical [61] and cyber-security [20], defenders are known to have knowledge about follower types, a classic case of *known-unknowns*. BSMGs provide the expressive power to represent this information; precisely θ_s represents the probability estimate with which a defender believes a certain kind of adversary is encountered in a particular state s of the game.

Note that a defender $\mathcal{D}$ is expected to deploy a system first. Thus, a strong threat model assumes that all the attacker types $\mathcal{A}_i$ know the defender's policy, making the Bayesian notion of Stackelberg Equilibrium an appropriate solution concept for such games. For a normal-form game, let a defender's mixed policy be denoted as x and let us denote an attacker type $\mathcal{A}_i$'s response set (i.e. a set of best responses to x) as $R^i(x)$. If the response set for all adversary types is singleton, then the action profile $(x, R^1(x), \ldots R^t(x))$ constitutes a Stackelberg Equilibrium of the normal-form game [60]. When the response set contains more than one action, the final response chosen can yield different rewards for $\mathcal{D}$. In such cases, a popular assumption made in general-sum games is to consider the response that results in the optimal rewards for $\mathcal{D}$; this is termed as the Strong Stackelberg Equilibrium (SSE) [75, 61, 118, 20]. In contrast to the notion of Weak Stackelberg Equilibrium, which considers the pessimistic case, an SSE is guaranteed to exist and yields a unique game value to the defender regardless

of the particular SSE chosen [57, 59]. Thus, we consider SSEs as the solution concept in BSMGs and highlight a few properties about player strategies at equilibrium.

Lemma 5.1.1. *For a given policy of the leader/defender in BSMG, every follower/attacker type will have a deterministic policy* $\forall s \in S$.

We note that BSMGs adhere to the Markovian nature and thus, a leader's policy is a Markov stationary policy [64]. For each follower type, a modified state transition function τ' that accounts for (1) the original transition τ and (2) the leader policy x constitutes a Markov Decision Process (MDP) (i.e. $\tau' = \tau \cdot x$). This guarantees that each follower type has a deterministic best-response policy given a leader's policy.

Corollary 5.1.1.1. *For an SSE policy of the defender, denoted as* x*, each attacker type* A_i *has a deterministic policy* q_i*. The action* $(x, q_1, \dots q_t)$ *denotes the SSE of the BSMG.*

We can now extend results known for Markov Games with a single follower type with a singleton response set where the distinction between Strong and Weak SE does not arise [127].

Lemma 5.1.2. *An action profile* $(x, q_1, \dots, q_t)$ *that yields the equilibrium values* $V^{\mathcal{D}}_{x,q}$ *and* $V^{\mathcal{A}_i}_{x,q}$ *to the players is at SSE of BSMG,* iff $\forall s \in S$ $(x(s), q_1(s), \dots, q_t(s))$ *is an SSE of the bi-matrix Bayesian game represented by the Q-values* $Q^{\mathcal{D}}_{x,q_i}(s), Q^{\mathcal{A}_i}_{x,q_i}(s)$.

First, note that a follower type can have a pure strategy response that corresponds to a Strong Stackelberg Equilibrium for single-stage normal-form games. Given a defender's policy, the attacker solves a linear (reward) maximization that always has a pure strategy in support of an optimal mixed strategy [61]. Hence, $q_i(s)$ is a pure-strategy of the bi-matrix game represented by the Q-values if state s. This ensures

that the SSE of the BSMG (Corollary 1) and the SSE of the bi-matrix games in each state admit pure-strategy for the individual follower types.

We will now prove the lemma in the forward direction by considering a proof by contradiction. Let us assume that (1) $(x, q_1, \ldots, q_t)$ is an SSE action profile of BSMG with equilibrium values V^p_{x,q_i} $\forall p \in P$ but, (2) $\exists s \in S$ for which $(x(s), q_1(s), \ldots, q_t(s))$ is not the SSE of the bi-matrix Bayesian game defined by the Q-values of s . If it were so, given that an SSE is bound to exist for the bi-matrix Bayesian game and it yields the highest unique pay-off to the players in a bi-matrix Bayesian game, a player p (D or A_i) should switch from their current strategy to an SSE in state s. This would clearly yield values higher than $V^p_{x,q_i}(s)$ for that state. This violates (1) because $V^p_{x,q_i}(s)$ was the equilibrium values of the BSMG corresponding to an SSE policy that has a unique optimal value. Thus, (1) implies (2).

A similar proof by contradiction can be constructed for the backward direction. Briefly, if the strategy in the states constitute SSE of the stage game but are not an SSE of the BSMG, it must be possible to switch the strategy in at least one state to yield the higher value guaranteed by the SSE of the BSMG. But if that is the case, the original assumption that the initial strategy in that state is an SSE is contradicted. □

When the parameters of a game are provided up-front, an approach similar to calculating Strong Stackelberg Equilibrium in Bayesian Games discussed in Chapter 3 alongside Mixed-Integer Non-Linear Programming approaches [64] or Bellman-style approaches for Markov Games discussed in Chapter 4 can be leveraged to find the defender's policy. In contrast, when game-parameters are difficult to provide upfront but interaction with an environment is considered possible, we can resort to reinforcement learning techniques. Before proposing our model-free multi-agent reinforcement

learning method in the next section, we briefly discuss how the various MTDs, used later in the experiments, are modeled as BSMGs.

5.1.1 Modeling Moving Target Defense Scenarios as BSMGs

In this section, we briefly highlight how the MTDs presented in the previous chapter can be modeled as a BSMG. As a quick recap, a Moving Target Defense (MTD) is defined by the tuple $\langle C, t, M \rangle$ where C represents the set of configurations a system can choose to be in, t represents a timing function that determines when a system switches and M represents a movement function that determines the movement policy. We will assume a constant function t for switching (the game clock) and discuss how we can leverage C to model the states and the actions of our BSMG. Finally, we leverage existing knowledge to model the follower types for the particular MTD scenario.

MTD for web-application security

In Chapter 3, we model an MTD for web-applications as a Bayesian Stackelberg Game (BSG). In addition, we consider the performance impact of movement between configurations (downtime, service latency, etc.) when coming up with a movement policy. Note that while the switching cost can't be modeled as the reward of the single-stage BSG, it can be modeled as a Markovian reward of a multi-stage game. By definition, this implies the formulation in Chapter 3 results in a state-agnostic sub-optimal policy. On the other hand, BSMGs can, given its sequential nature, express (and as we will later show, learn) this information.

The BSMG has $|C|$ states, each representing a configuration of the MTD system. Each configuration has an equal probability of being the start state in an episode and

there exists no terminal state. In each state, $s \in S, A^{\mathcal{D}}(s) = C$, i.e. the defender can choose to move to any configuration (this includes remaining in the same state). The three attacker types are denoted as $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3\}$ and the probability distribution, provided in [20], over these types in-state s represented is θ_s. The adversary's action sets, denoted by $(A^{\mathcal{A}_1}, A^{\mathcal{A}_2}, A^{\mathcal{A}_3})$, as presented in Chapter 3, represent mined CVEs from the National Vulnerability Database [128] and the distribution over attacker types remain the same in all states of the BSMG.

MTD against multi-stage attacks

While BSMGs allow us to represent uncertainty over attacker types, game-theoretic formulations of MTD in cloud networks discussed in Chapter 4 consider single follower types. Thus, they can be simply boiled down to a Markov Game formulated in Section 4.2. We note that MGs are a special case of our proposed BSMGs (see Figure 5.5) and can incorporating ongoing research on the characterizing attacker types in the context of such scenarios [111].

5.2 Strong Stackelberg Q-learning in BSMGs

While game-theoretic formalism has been used to model various cyber-security scenarios [22, 13, 20], it is impractical to expect security experts to provide the parameters of the game upfront [2, 55, 56]. In the context of Moving Target Defense (MTD) in particular, determining the impact of various attacks, the asymmetric impacts of a particular defense on performance, and the switching cost of a system are better obtained via interaction with an environment. Further, there exists uncertainty regarding the success of an attack (eg. a buffer overflow attack may need significant tinkering for it to be successful) and also the success of defense mechanisms (eg. In-

Algorithm 3 Bayesian Strong Stackelberg Q-learning (BSS-Q) for BSMGs.

In: $(P, S, A, \Theta, \gamma)$, `mtd_sim`, α, num_episodes

Out: Policies of the players x for $\mathcal{D}$, $q^i \ \forall i \in \mathcal{A}$

while num_episodes > 0 **do**

  $s \leftarrow$ sample start state from S

  while $s \neq$ terminal state OR !max_eps_len **do**

    $i \leftarrow$ sample attacker type using θ_s from A

    $a^{\mathcal{D}}, a^{\mathcal{A}_i} \leftarrow \epsilon$-greedy sampling form $x(s)$, $q_i(s)$

    $r^{\mathcal{D}}, r^{\mathcal{A}_i}, s' \leftarrow$ `mtd_sim`.act(s, a^D, a^A)

    $Q^{\mathcal{D},i} \leftarrow (1-\alpha)Q^{\mathcal{D},i}(s, a^{\mathcal{D}}, a^{\mathcal{A}_i}) + \alpha[r^{\mathcal{D}} + \gamma^{\mathcal{D}} V^{\mathcal{D}}(s')]$

    $Q^{\mathcal{A}_i} \leftarrow (1-\alpha)Q^{\mathcal{A}_i}(s, a^{\mathcal{D}}, a^{\mathcal{A}_i}) + \alpha[r^{\mathcal{A}_i} + \gamma^{\mathcal{D}} V^{\mathcal{A}_i}(s')]$

    $(x, q_j), (V^{\mathcal{D}}, V^{\mathcal{A}_j}) \leftarrow$ solve Bayesian Stackelberg Game$(Q^{\mathcal{D},i}, Q^{\mathcal{A}_j}) \ \forall j$

  end while

end while

trusion Detection Systems based on machine learning can be inaccurate) which can be better inferred via repeated interactions.

In existing works on MTD, the goal, in the presence of (1) game parameters and (2) incomplete information about an adversary, the goal is to learn a robust policy that works best, in expectation, against all adversaries. When modeled as a multi-agent reinforcement learning problem, an interesting distinction ensues. If the defender gets to interact with the environment and an actual adversary, they can update their incomplete information about the adversary, leading to a Bayesian style update regarding the attacker types after every interaction [54]. Unfortunately, reinforcement learning methods are often sample-inefficient and require abundant interaction with a real-world adversary. In cyber-security scenarios, this is nearly

impossible. Thus, the best the defender can do is to simulate an adversary in their head and learn a robust policy via interaction with the environment.

To learn a robust policy, we consider the use of a Multi-agent Reinforcement Learning approach for BSMG. Specifically, given the inherent leader-follower paradigm present in our setting, we propose the use of a Bayesian Strong Stackelberg Q-learning (BSS-Q) approach for BSMGs discussed in Algorithm 3. The approach is similar to existing work in multi-agent reinforcement learning (MARL) and considers a Bellman-style Q-learning approach for calculating the agent policies over time. In lines 9 and 10, we update the Q-values for the players $\mathcal{D}$ and adversary type $\mathcal{A}_i$ in the state s using the rewards obtained by acting in the environment. Since we simulate the adversary, we can select an action on its behalf and send it across to the simulator `mtd_sim`. Given `mtd_sim` which has an idea whether the attack succeeded or failed, it can send us back the attacker's reward. [1] In existing works on MARL, they make a default assumption that the defender gets the attacker's reward even when the adversary is not simulated [127]. While this assumption is somewhat justified in the context of constant-sum or common-payoff games, it becomes unrealistic in the context of general-sum games.

In line 11, we use a Bayesian Stackelberg Game solver to calculate the BSS of the normal form Bayesian bi-matrix game defined by the Q-values in state s. It is known that finding an SSE of a Bayesian Stackelberg Game is NP-hard and thus, the computation in line 10 might seem prohibitive. In practice, as shown in our experiments, compact representation of the scenario as a MILP [61] can help in computing the value and the policy within a second even for web-application domains with more than 300 executable attack actions [20]. Note that even though only one follower

[1] If interaction with an adversary is possible, the defender should consider a Bayesian style update of the parameters θ_s depending on the observed action and the observed reward after line 10.

type acts in the environment, the change in the defender's policy because of this can lead to the other follower types switching their actions. Hence, solving the Bayesian stage game (in its full glory) becomes essential to converge to an SSE policy of the BSMG (and batch update methods, which can speed-up computation, become less reliable). Methods that scale-up equilibrium computation in security games [118] rely on a known reward structure. Unfortunately, this makes it difficult to justify their use in our setting, where the rewards are unknown and thus, may have an arbitrary reward structure.

Proposition 1. (Convergence Result) *The Bayesian Strong Stackelberg Q-learning approach converges to the BSS of a BSMG.*

Our proof of convergence is inspired from the initial work by [129]. Let us call the Q-value update step as a process $\Omega : \mathbf{Q} \rightarrow \mathbf{Q}$ where $\mathbf{Q}$ represents the space of Q-values. Formally we can express the update equation for the leader $\mathcal{D}$ as,

$$\Omega(Q^{\mathcal{D},i}(s, a^{\mathcal{D}}, a^{\mathcal{A}_i})) \;\; = \;\; U^{\mathcal{D},i}(s, a^{\mathcal{D}}, a^{\mathcal{A}_i}) + \gamma V^{\mathcal{D}}(s_{T+1})$$

Where $V^{\mathcal{D}}$ represents the leader's game value in the Bayesian Stackelberg Game (BSG) defined by Q-values and the distribution over follower types in state s_{T+1}. With some abuse of notation, we can drop the arguments $(s, a^{\mathcal{D}}, a^{\mathcal{A}_i})$ for both the functions $Q^{\mathcal{D}}$ and $U^{\mathcal{D}}$ and rewrite the above equation by expanding the value function as follows.

$$\Omega(Q^{\mathcal{D},i}) = U^{\mathcal{D},i} + \gamma V^{\mathcal{D}}(s_{T+1})$$

To prove convergence of the function/operator/process Ω, we need to show the following two conditions hold (as described in [129]; the other two conditions mentioned hold trivially in our setting, as it holds in the case of other Q-learning approaches).

(1) The following processes converge to a fixed point.

$$\begin{aligned} Q_{T+1}^{\mathcal{D},i} &= (1-\alpha_T)Q_T^{\mathcal{D},i} + \alpha_T \Omega(Q_{sse}^{\mathcal{D},i}) \ \forall \ i \\ Q_{T+1}^{\mathcal{A}_i} &= (1-\alpha_T)Q_T^{\mathcal{A}_i} + \alpha_T \Omega(Q_{sse}^{\mathcal{A}_i}) \ \forall \ i \end{aligned}$$

where Q_{sse} represents the Q-values at SSE of the BSMG.

(2) The process Ω is a real contraction operator.

$$||\Omega(Q) - \Omega(\bar{Q})|| \leq a||Q - \bar{Q}|| \quad \forall Q, \bar{Q} \in \mathbf{Q}$$

where $0 < a < 1$ and $||\cdot||$ denotes the supremum operator over the vector space Q.

To prove the conditions in (1), we leverage the Condition Averaging Lemma stated in [129] and thus, have to show that,

$$Q_{sse}^{\mathcal{D},i} = \mathbb{E}[\Omega(Q_{sse}^{\mathcal{D},i})] \quad , \quad Q_{sse}^{\mathcal{A}_i} = \mathbb{E}[\Omega(Q_{sse}^{\mathcal{A}_i})] \ \forall \ i$$

where the expectation is over the states reached. To show this, we first expand the right hand side of the equation and showing that this expansion is equal to the left hand side.

$$\begin{aligned} \mathbb{E}[\Omega(Q_{sse}^{\mathcal{D},i})] &= U^{\mathcal{D}} + \gamma \ \mathbb{E}[V^{\mathcal{D}}(s')] \\ &= U^{\mathcal{D}} + \gamma \ \mathbb{E}[V_{sse}^{\mathcal{D}}(s')] \\ &= U^{\mathcal{D}} + \gamma \ \sum_{s'} \tau(s'|s,\sigma) * V_{sse}^{\mathcal{D}}(s') \\ &= Q_{sse}^{\mathcal{D},i} \end{aligned}$$

where $V_{sse}^{\mathcal{D}}$ indicates the game value of the defender at SSE. $V^{\mathcal{D}} = V_{sse}^{\mathcal{D}}$ because the Q-matrices in all state represent the value at SSE. The final equality is a result of

the Bellman equation for multi-agent settings. It is easy to see that the same line of reasoning holds for all all follower types.

To prove the condition in (2), we will first expand the Left Hand Side (LHS) followed by the expansion of the Right Hand Side (RHS). Then we will show that a stricter case of the inequality is satisfied. We first show this for the follower and then, for the leader.

$$
\begin{aligned}
& ||\Omega(Q^{\mathcal{A}_i}) - \Omega(\bar{Q}^{\mathcal{A}_i})|| \\
= \; & \max_s \big(\Omega(Q^{\mathcal{A}_i}(s, x, R(x))) - \Omega(\bar{Q}^{\mathcal{A}_i} i(s, x, R(x)))\big) \\
= \; & \gamma \max_s \big(V^{\mathcal{A}_i}(s') - \bar{V}^{\mathcal{A}_i}(s')\big) \\
= \; & \gamma \max_s \big(\max_x Q^{\mathcal{A}_i}(s', x, R^{\mathcal{A}_i}(x)) - \max_x \bar{Q}^{\mathcal{A}_i}(s', x, R^{\mathcal{A}_i}(x))\big) \\
= \; & \gamma\big(\max_x Q^{\mathcal{A}_i}(s', x, R^{\mathcal{A}_i}(x)) - \max_x \bar{Q}^{\mathcal{A}_i}(s', x, R^{\mathcal{A}_i}(x))\big) \qquad (5.1)
\end{aligned}
$$

The first equality is based on the use of the *supremum* operator. Given that the max occurs for some s, without loss of generality, we can assume this state is s going to state s'. In a similar way, we can expand the RHS for the follower.

$$
\begin{aligned}
& a||Q^{\mathcal{A}_i} - \bar{Q}^{\mathcal{A}_i}|| \\
= \; & a \max_s \max_x \max_q \big(Q^{\mathcal{A}_i}(s, x, q) - \bar{Q}^{\mathcal{A}_i}(s, x, q)\big) \\
\geq \; & a \max_x \max_q \big(Q^{\mathcal{A}_i}(s', x, q) - \bar{Q}^{\mathcal{A}_i}(s', x, q)\big) \\
\geq \; & a \max_x \big(Q^{\mathcal{A}_i}(s', x, R^{\mathcal{A}_i}(x)) - \bar{Q}^{\mathcal{A}_i}(s', x, R^{\mathcal{A}_i}(x))\big) \qquad (5.2)
\end{aligned}
$$

Note the we now have a stricter version of the RHS. If we can now show that Equation 5.1 $\leq$ Equation 5.2, then we can prove condition (2) holds for the Q-values of all follower types. Given $0 \geq \gamma < 1$, we can consider $a = \gamma$. Now we have,

$$
\begin{aligned}
& \gamma\big(\max_x Q^{\mathcal{A}_i}(s', x, R^{\mathcal{A}_i}(x)) - \max_x \bar{Q}^{\mathcal{A}_i}(s', x, R^{\mathcal{A}_i}(x))\big) \\
= \; & a\big(\max_x Q^{\mathcal{A}_i}(s', x, R^{\mathcal{A}_i}(x)) - \max_x \bar{Q}^{\mathcal{A}_i}(s', x, R^{\mathcal{A}_i}(x))\big)
\end{aligned}
$$

$$
\begin{aligned}
&\leq a \max_x \left(Q^{\mathcal{A}_i}(s', x, R^{\mathcal{A}_i}(x)) - \bar{Q}^{\mathcal{A}_i}(s', x, R^{\mathcal{A}_i}(x))\right) \\
&\leq a||Q^{\mathcal{A}_i} - \bar{Q}^{\mathcal{A}_i}||
\end{aligned}
$$

The first inequality holds because in the second step, one can select two different x-s to minimize the difference while in the third step, one is constrained to select the same x for both the Q-values.

Now, we show that Ω is also a contraction operator for the Q-values of the defender. Showing the property holds is difficult to show for individual follower types because of the Bayesian nature of the game. It is possible to show this for a transformed attacker conjured using the Harsanyi transformation [130]. In this setting, the single attacker type has actions that are the cross product of the action of all other players and the utilities of the Q-value matrix represent the expected Q-value over the original attacker types. Given this single attacker type, we use $Q^{\mathcal{D}}$ to denote the Q-values against this newly transformed attacker. Given the solver we are using in our BSS Q-learning approach in calculating SSE of the BSG stage games is equivalent to the SSE of this transformed game [61], showing Ω is a contraction operator for $Q^{\mathcal{D}}$ is sufficient to show convergence. We use $\mathcal{A}$ in the superscripts to denote value for this transformed attacker type.

$$
\begin{aligned}
&||\Omega(Q^{\mathcal{D}}) - \Omega(\bar{Q}^{\mathcal{D}})|| \\
&= \max_s \left(\Omega(Q^{\mathcal{D}}(s, x, R^{\mathcal{A}}(x))) - \Omega(\bar{Q}^{\mathcal{D}}(s, x, R^{\mathcal{A}}(x)))\right) \\
&= \gamma \max_s \left(V^{\mathcal{D}}(s') - \bar{V}^{\mathcal{D}}(s')\right) \\
&= \gamma \max_s \left(\max_x \sum_i \theta^i(s') Q^{\mathcal{D},i}(s', x, R^{\mathcal{D},i}(x)) - \max_x \sum_i \theta^i(s') \bar{Q}^{\mathcal{D},i}(s', x, R^{\mathcal{D},i}(x))\right) \\
&= \gamma\left(\max_x \sum_i \theta^i(s') Q^{\mathcal{D},i}(s', x, R^{\mathcal{D},i}(x)) - \max_x \sum_i \theta^i(s') \bar{Q}^{\mathcal{D},i}(s', x, R^{\mathcal{D},i}(x))\right)
\end{aligned}
$$

$$
\begin{aligned}
&= a\big(\max_x \sum_i \theta^i(s') Q^{\mathcal{D},i}(s', x, R^{\mathcal{D},i}(x)) - \max_x \sum_i \theta^i(s') \bar{Q}^{\mathcal{D},i}(s', x, R^{\mathcal{D},i}(x))\big) \\
&\leq a \max_x \big(\sum_i \theta^i(s') Q^{\mathcal{D},i}(s', x, R^{\mathcal{D},i}(x)) - \sum_i \theta^i(s') \bar{Q}^{\mathcal{D},i}(s', x, R^{\mathcal{D},i}(x))\big) \\
&\leq a \max_s \max_x \big(\sum_i \theta^i(s) Q^{\mathcal{D},i}(s, x, R^{\mathcal{D},i}(x)) - \sum_i \theta^i(s) \bar{Q}^{\mathcal{D},i}(s, x, R^{\mathcal{D},i}(x))\big) \\
&\leq a \max_s \max_x \Pi \max_{q^{\mathcal{A}_i}} \big(\sum_i \theta^i(s) Q^{\mathcal{D},i}(s, x, q^{\mathcal{A}_i}) - \sum_i \theta^i(s) \bar{Q}^{\mathcal{D},i}(s, x, q^{\mathcal{A}_i})\big) \\
&\leq a \max_s \max_x \max_{q^{\mathcal{A}}} \big(Q^{\mathcal{D}}(s, x, q^{\mathcal{A}}) - \bar{Q}^{\mathcal{D}}(s, x, q^{\mathcal{A}})\big) \\
&= a||Q^{\mathcal{D}} - \bar{Q}^{\mathcal{D}}||
\end{aligned}
$$

If we were now to consider $\bar{Q} = Q_{sse}$ for all the player and player types, then the Q-values learned by our method will approach Q_{sse}. While this completes our convergence proof, we note that the convergence rate depends on two factors. First, Selecting randomly among best-responses, even if multiple exist, for the follower results in slower convergence. Selecting consistently in some order (eg. first after sorting the response set) results in faster convergence. Note that random selection does not cause issues beyond slowing down convergence because for each follower type, given a leader's strategy, regardless of the best response strategy selected, the game value for both players remains the same due to the nature of SSE [57, 59]. Second, a similar line of reasoning for the defender concludes that a pre-defined selection mechanism can result in faster convergence. □

5.3 Experiments

We conduct experiments to understand the effectiveness of the learned movement policies for the two MTD scenarios described above. As many existing baselines can't handle unknown utilities and transition dynamics, we develop an OpenAI style [125]

Agents	Time (sec)
Static	84.097 ± 0.157
B-EXP-Q	224.827 ± 1.449
BSS-Q	151.127 ± 13.403
B-Nash-Q	> 3600

Table 5.1: Time taken by the different agents.

game simulator, similar to other works in reinforcement learning for cyber-security [55, 56], that are aware of the underlying game parameters but interacts with the learning agents only via selected public-facing APIs. This helps us to compare against baselines that assume game parameters are available. While the impacts of attack actions are obtained in the simulator using the Common Vulnerability Scoring Service (CVSS), we can consider real system interaction, pending investigation, to obtain less-informative and sparse rewards. More details about the game simulator and additional experiments can be found in Appendix B. For both experiments, the defender (who samples a follower type in each interaction) is a single-thread process and regardless of their policy, are pitted against a strategic and ration adversary. The code used Gurobi for solving the Bayesian Stackelberg Game (BSG) game in line 11 of Algorithm 3 and ran on an Intel Xeon(R) CPU E5-2643 v3 @ 3.40GHz with 64GB RAM.

5.3.1 MTD for Web-applications

We use the game parameters obtained in Chapter 3 to design our game simulator. The simulator, given the current state and the defender's action, lets us return the switching costs as a part of the reward obtained via interaction with the environment. The game has four states, each representing a full-stack configuration of the system–

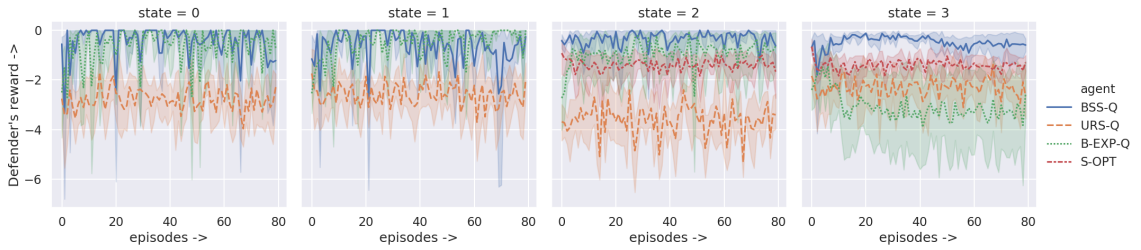


Figure 5.2: The defender's value in the four states of the BSMG modeling for the web-applications MTD when using BSS Q-learning (blue) compared to Uniform Random Strategy (orange) [1], a Bayesian version of EXP-Q learning (green) [2] and the optimal stage-agnostic strategy (red) inferred in Chapter 3.

{(py, MySQL),(py, PostgreSQL),(Php, MySQL),(Php, PostgreSQL))}. We consider three attacker types– $\mathcal{A}_1$ (database hacker) with 269 actions, $\mathcal{A}_2$ (script kiddie) with 34 actions, and $\mathcal{A}_3$ (mainstream hacker) with 48 actions.

The defender's policy in each state is a mixed strategy that directs how to switch to a different configuration, while an attacker type's policy prioritizes attacks that cause maximum damage. In Figure 5.2, we plot the defender's reward (over 6 trials) in this BSMG for our BSS-Q learning agent and other baselines. In each setting, we use a discount factor of $\gamma = 0.8$, an exploration rate of $\epsilon = 0.1$ (that gradually decays to 0.05) and initiate the agents with a uniform random strategy (except in the case of S-OPT explained as follow). The average time used by the methods for one trial is shown in Table 5.1. We will now explain all the baselines considered.

Inferred Movement Policies These defense policies, evident from its name, are determined up-front (using game parameters provided initially) and do not change. The placebo baseline, used as a sanity check in the context of MTDs, is the Uniform Random Strategy (URS); it selects each action in a state with equal probability

Figure 5.3: Comparing BSS-Q learning agents with URS (orange) and the EXP-Q learning (green) on MTD for web-applications when switching costs are represented in the transition function of BSMGs.

[1]. Then, we consider the state-agnostic optimal policy (S-OPT) determined in Chapter 3.

Learning Agents In [2], the authors leverage adversarial multi-arm bandits in learning policies for an agent in Markov Security Games. While the paper draws inspiration from works in Stackelberg Security Games [118], the EXP Q-learning approach does not consider (1) a strategic adversary that can adapt or (2) uncertainty over attacker types. We adapt their algorithm for BSMGs by ensuring that the update to the sum of rewards is weighed by the attacker type's probability. We call this the Bayesian EXP Q-learning agent (B-EXP-Q). The Bayesian Nash Q-learning (B-Nash-Q) [54], even after we remove the Bayesian update of θ_s, does not scale for this domain and thus, can only be compared against in simple toy domains (a non-Bayesian version is discussed for the other MTD).

In Figure 5.2, we see that the BSS Q-learning agent outperforms URS in all the states of the BSMG and better than B-EXP-Q and S-OPT in s_2 and s_3, attaining a reward close to 0, which is the maximum reward possible in this game. In s_0 and s_1, the B-EXP-Q agent, similar to the BSS-Q agent, converges to an optimal

movement strategy. Meanwhile, in s_3, the best response of the attacker results in lower rewards for a particular defense action, in turn making this action less probable. As soon as the defender switches to a more promising action, the attacker changes their response to yield a low value for that action. It should be no surprise that defense strategies learned via single-agent RL methods (eg. [2, 55, 56]) are prone to be exploitable against strategic opponents in cyber-security scenarios. The existence of such a cycle makes the learned policy exploitable, resulting in low rewards consistently. In such cases, even the URS yields better rewards because its lack of bias makes it less exploitable. The S-OPT, as described in [20], yields a strategy that moves between two MTD configurations represented by s_2 and s_3. As such a strategy can never find itself in any other state of the game, to be fair, we ensure that the start state in each episode is uniformly sampled from s_2, s_3. Thus S-OPT has no footprint in the states s_0 and s_1. As stated before, the S-OPT strategy, a resultant of the state-agnostic game-theoretic formulation, is doomed to be sub-optimal. Unsurprisingly, the policy learned by BSS-Q is shown to be better in s_2 and s_3. A drop in rewards near episode 70 in s_1 for BSS-Q can be attributed to the discovery of powerful attacks by two follower types in one trail.

Switching Cost as Transition Dynamics We consider a different perspective on switching costs that is possible to represent using BSMG but cannot be captured by our model presented in Chapter 3. In this setting, we do not capture the switching cost as part of the reward metric, but use it to guide the transition dynamics of the underlying environment. Precisely, when an `act` API is called, we use the defender's action $a_{\mathcal{D}}$ to perform the switch. While performing the switch action, if we observe a drop in the successful processing of packets or an increase in the response latency, calibrated by a threshold, we abort the move action and stay in the same state. This

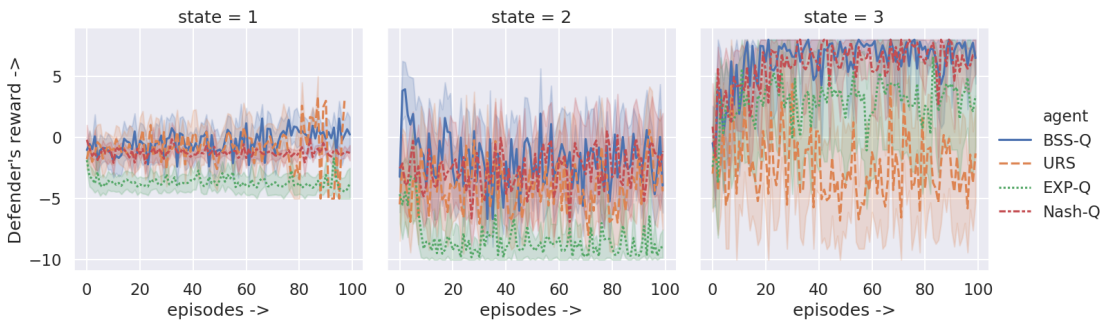


Figure 5.4: Values in the BSMG-based MTD for IDS placement when using BSS Q-learning (blue) compared to Uniform Random Strategy (orange), the EXP-Q learning (green) and the Nash Q-learning (red).

makes the underlying environment stochastic, which as per the model in Chapter 3, was deterministic. We use the existing switching cost to guide the transition dynamics, i.e. expensive switches have a higher probability of failing to execute the switch and thus, remain in the same state.

In Figure 5.3, we plot the rewards obtained by the proposed BSS Q-learning agent in comparison to the baselines described in the paper– namely the Uniform Random Strategy (URS) and the adversarial multi-arm bandit based EXP-Q learning agent. The state-agnostic optimal policy is ill-defined in this scenario as it only expects to find itself in two configurations of the system and such assurances are impossible given the stochastic nature of the environment. We use a discount factor of 0.8, an exploration rate of 0.15 (that decays gradually to 0.05, and a learning rate of 0.06.

As before, BSS-Q gathers higher reward, over six trails, than URS. In this setting, it performs better than EXP-Q in three states s_0, s_2 and s_3 (the margin of improvement being significantly better in s_0 and s_2).

5.3.2 MTD for IDS Placement

We consider the General-Sum Markov Game formulated in Chapter 4. As described earlier, this domain has four states, of which, one is a terminal state. As the set of follower types in this game in singleton, the BSMG model boils down to Markov Game. Hence, the vanilla EXP-Q agent can be used (albeit against a rational follower). The URS baseline remains unchanged, S-OPT deems to exist and instead of Bayesian Nash, we can consider the relatively more scalable Nash Q-learning (Nash-Q) agent [131]. The rewards obtained by the various agents in the three non-terminal states of the game are plotted in Figure 5.4. Given the domain has relatively fewer actions, we average the reward over 10 trials; each trial had 100 episodes and with an exploration rate of 0.1 and a discount factor of 0.8.

The BSS Q-learning agent outperforms the two previous baselines– URS and EXP-Q– in at least one of the three states. In this setting, the Stackelberg threat model adversely affects the policy learned by EXP-Q resulting in consistent low rewards for states s_1 and s_2. While we showed in Section 4.2 that the SSE $\subseteq$ NE for the MG devised in this setting, owing to the structure of the defender's strategy sets, we see that Nash-Q performs slightly worse the BSS-Q in state s_1. This happens because of the existence of multiple Nash Equilibria (see Section 2.2); the strategy found by Nash-Q Learning does not guarantee the optimal reward [132].

5.4 Related Work

Multi-agent Reinforcement Learning in Markov Games A standard solution strategy in MGs, when τ and U are unknown but a simulator is available, is to adapt the Bellman's update used in the single-agent Markov Decision Processes (MDPs) for

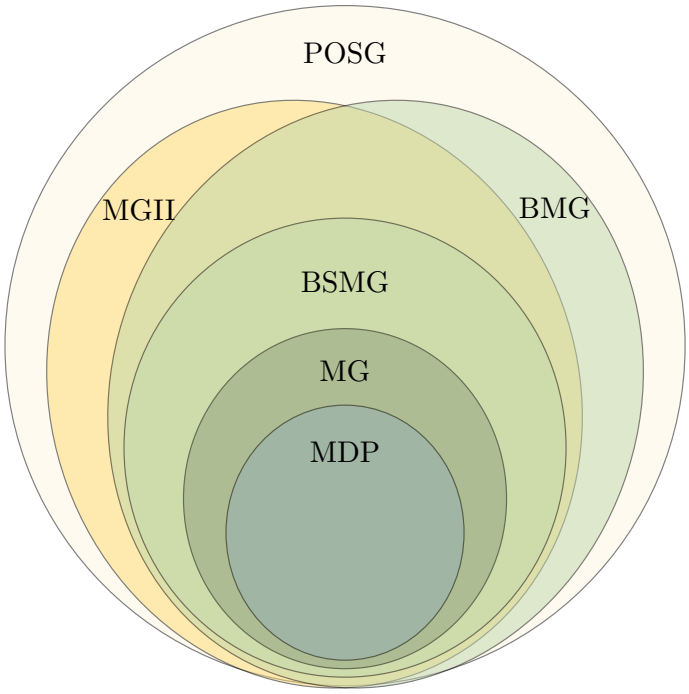


Figure 5.5: Situating Bayesian Stackelberg Markov Games (BSMGs) in the context of other game-theoretic models that try to capture incomplete information in Stochastic Games.

multi-agent reinforcement learning to learn equilibrium policies [132]. In the context of MARL, researchers have investigated different notions of equilibrium. The min-max Q-learning is meaningful when the game has a zero-sum reward structure [107]. On the other hand, Nash Q-learning, introduced in [131], has been categorized into two types by [133]– Friend, where the game defined by the Q-values always allows for an optimal joint action profile, and Foe, where the game admits a saddle point solution. The convergence of these algorithms is mostly shown by the fact that Q-values of the states in self-play, given infinite exploration, approach the *correct* Q-values (i.e. calculated Q-values if all the parameters of the game were provided upfront). The convergence of Nash Q-learning in the context of general-sum games becomes

difficult because of the existence of multiple equilibria and the lack of a common incentive or co-ordination amidst agents [132]. In correlated equilibrium (CE) Q-learning [134], authors assume the existence of a correlation device accessible to both players. However, the authors show that the learned strategies of the players converge to an uncorrelated equilibrium. On the other hand, in Stackelberg Q-learning [127], there exists a leader-follower paradigm among the players, i.e .the leader's strategy can be observed by a follower before the latter commits to an action. Convergence guarantees in self-play, although popular, become less meaningful when action sets of the players are different (defense configuration *vs.* exploits) and game utilities have a general-sum reward structure.

The Landscape of Existing Games We seek to answer two questions– where does our BSMG fit in and why it is useful. In Figure 5.5, we graphically situate BSMG in the landscape of existing work. BSMG, as seen in the experiments, can generalize Markov Games [110] (and therefore, MDP). Instead of assuming infinite reasoning capabilities required for Bayesian Nash equilibrium [135], Bayesian Markov Game considers scenarios where players have finite levels of belief about other player(s) [136]. On the other hand, Markov Games with Imperfect Information (MGIIs) assumes a Markovian property over the state, the observations, and the joint set of actions; this results in reasoning over opponent types and allows them to decompose Partially Observable Stochastic Games (POSGs) [137] into a set of Bayesian stage-games [138]. In BSMGs, the assumption of a pre-specified distribution over attacker types helps us (1) avoid reasoning over the nested belief space, and (2) can be interpreted as the private information of the opponent in MGIIs (and POSGs) as being provided upfront. Thus, BSMG becomes a special case of MGII and BMG with the added semantics of leader-follower interaction. Our assumptions (about modeling im-

perfect information) helps our game be scalable while providing adequate expressive power in the context of MTDs.

Leader-follower scenarios Researchers have investigated solution concepts in the context of Stochastic games with multiple-followers, but did not model the aspect of incomplete information in them. [2] The interaction is generally modeled as a Semi-Markov Decision Process [139] and the improvements consider (1) multiple followers [140], (2) factored state spaces [141], (3) methods based on deep reinforcement learning [142, 143] etc.

Reinforcement Learning in MTDs Works that are precursors to our BSS-Q learning approach are the min-max Q-learning [53] in the context of complete information MGs and the Bayesian Nash Q-learning [54] for dynamic placement of sensors. Recent works that model the multi-agent cyber-scenarios as an MDPs (RL in Flip-it games [55]) or POMDPs (RL for MTDs [56]), as shown above, can generate policies that can be exploited by a strategic adversary.

5.5 Concluding Remarks

This chapter took a critical view of the different game-theoretic models developed for MTDs in web-application and cloud network security in the previous chapters. We proposed a novel game– the Bayesian Stackelberg Markov Game (BSMG)– that considers the leader-follower scenario and uncertainty over adversary types in Markov Games. We showed that BSMG is a unified framework to characterize optimal movement policies in Moving Target Defenses (MTDs).

[2] Extending the uncertainty over follower-types in BSMGs to multiple followers (with their types) can result in further challenges for scalability.

We then highlighted the challenge of being provided the game parameters up-front. This makes the inference mechanisms devised in Chapter 3 and 4 ineffective. To address this, we consider that a simulation environment, a stand-in for the real cybersecurity environment is available and proposed a Bayesian Strong Stackelberg Q-learning (BSS-Q) approach to learning robust policies. We showed that BSS-Q learning converges to the SSE of the BSMG. Experiments conducted in the cyber-security scenarios presented earlier– MTD for web-application and MTD for cloud-networks– showed that policies learned using BSS-Q outperform existing baselines.

Part II

Moving Target Defense for all (MTD for all)

Chapter 6

MOVING TARGET DEFENSE FOR CYBER-DECEPTION

Table of Contents ◇ 1 ◇ 2 ◇ 3 ◇ 4 ◇ 5 ◇ 6 ◇ 7 ◇ 8 ◇ 9

C	Deception Surface Shifting using Conflict-free Honey-patches
t	Constant/Fixed Time Period
M	Stackelberg Strategy of a Bayesian Normal-form Game

The second part of this thesis considers the novel use of Moving Target Defense in existing application domains. Before we move away from cyber-systems and delve into MTDs for machine learning systems and cyber-physical systems in the next chapters, we unite the two most promising proactive defenses in cyber-security– (1) cyber deception [4] and (2) moving target defense [5] in this chapter.

To anticipate and thwart directed cyberattacks, deceptive *honey-patching* [144] has been proposed as a language-based methodology to patch software security vulnerabilities in a way that future attempted exploits of the patched vulnerability appear successful to attackers even when they are not. Such techniques can mask patching lapses, impeding attackers from discerning which systems are genuinely vulnerable, and confuse attackers about which patched systems masquerade as unpatched systems. Detected attacks are surreptitiously redirected to isolated, unpatched decoy environments with the full interactive power of the targeted victim server. The de-

coy environments dis-inform adversaries with honey-data and aggressively monitor adversarial behavior.

While these capabilities offer potentially promising defense layers against determined adversaries skilled at evading traditional honeypots, the question of which vulnerabilities to deceptively emulate or how to automatically evolve the deceptive attack surface with evolving adversarial Tools, Techniques and Procedures (TTP) has remained relatively unstudied. For example, Chameleon [145], Honeyd scripts [146], and Cloxy [147] deployments all presently rely upon the human selection of vulnerabilities and versions, which is sub-optimal for waging long-term deceptive campaigns, because it can result in deceptions becoming predictable and stale, potentially affording attackers the opportunity and time to fingerprint and circumvent deceptive applications. As new vulnerabilities emerge, attacks and probing activity change [148, 149, 150, 151], potentially rendering old deceptions less enticing to cybercriminals.

To overcome this disadvantage, this chapter proposes *software deception steering* that proposed moving target defense techniques for counter reconnaissance and attack intelligence gathering. We leverage application-level, deceptive attack responses through honey-patching to continuously adapt the *deception surface* of the target application. To this end, we designed and implemented QUICKSAND, an adaptive software version emulation architecture, in which the set of fake vulnerabilities is dynamically re-selected to increase the likelihood of deceiving and entrapping attackers. Based on the vulnerability context (e.g., vulnerability risk scores, attack history), QUICKSAND chooses to emulate a particular software version with a particular set of (fake) vulnerabilities. This moving deception surface undermines the attacker's ability to identify and detect deceptions, and increases the likelihood of gathering *high-quality* threat data reflective of advanced attacks by skilled adversaries (rather

than merely well-known attacks by unskilled adversaries, for which threat data is less useful). To summarize, our work makes the following contributions.

- We propose a moving target approach to cyber-deception that dynamically honey-patches software, rendering it less predictable and more robust against attackers' anti-deception efforts.
- We propose, design, and implement an effective version-control strategy to facilitate patch re-selection and automatically resolve source-level conflicts between patches.
- We model the software emulation process as a Bayesian Stackelberg Game (BSG), similar to the one described in Chapter 3, to infer effective movement strategies that account for pragmatic aspects of deception, including the utility of intelligence-gathering actions, impact of vulnerabilities, cost of patch deployment, the complexity of exploits, and the attacker model.

6.1 Overview

First, we outline our new moving deception maneuver, followed by primary challenges and corresponding design decisions for software deception steering. Finally, we summarize background literature and our threat model.

6.1.1 Software Deception Steering

We define *deception steering* as the use of cyber deception for manipulating adversaries into revealing threat intelligence useful for detecting and thwarting future attacks. Leveraging vulnerability metadata and intrusion alerts collected at the network perimeter, QUICKSAND dynamically adapts the target application to emulate a

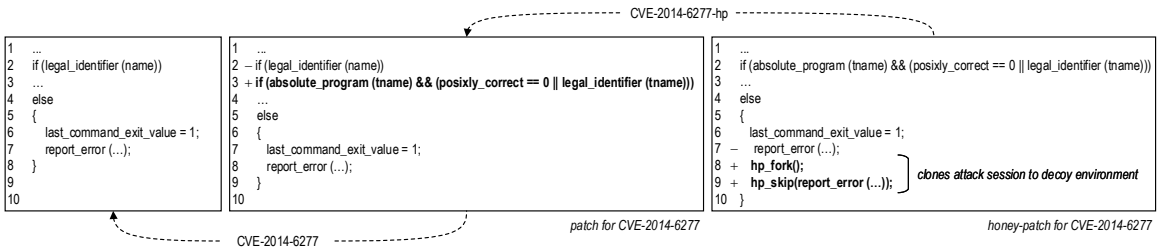


Figure 6.1: Patch and honey-patch for CVE-2014-6277 (abbreviated), and dependencies between them denoted by dashed arrows

particular software version, with a particular set of honey-patched vulnerabilities (and all other known vulnerabilities regular-patched), a particular set of features/modules enabled, and a particular guest OS version in decoys.

The scope of adaptation can go beyond the application and host boundaries; for instance, perimeter defenses (if any) can also be reconfigured to intentionally allow previously filtered attacks to reach the honey-patch. This reconfiguration need not happen live: it can be re-selected during nightly reboots, for example. The selections are based on which configuration is likely to gather the most useful threat data given the history of past attacks.

6.1.2 Design Principles

Software deception steering requires a patch management framework that facilitates software version composition and minimizes source code-level conflicts between patches. QUICKSAND defines honey-patches as modifications to their corresponding regular, vendor-supplied patches. For instance, Figure 6.1 exemplifies a vulnerability causing the GNU Bash shell to improperly parse function definitions in the values of environment variables [152]. Before the patch, the vulnerable shell interpreter allowed remote attackers to execute arbitrary code or cause a denial of service on the victim's

machine. The patch, named CVE-2014-6277 in this example, fixes the vulnerability by extending the check for what constitutes a legal function identifier to include some extra sanity checks (Lines 2–3 in the patch code, depicted in `diff` style). The honey-patch CVE-2014-6277-hp modifies the original patch to fork attacks onto decoy environments while impersonating the unpatched code (Lines 8–9 in the honey-patch code) to deceive adversaries. Encoding honey-patches in this manner naturally models the dependency among honey-patches, their corresponding patches, and unpatched source code. It also makes patch/honey-patch pairs conflict-free by construction, greatly simplifying the task of composing new versions of the target application.

Patch dependencies (denoted by dashed arrows between patches) are calculated based on how patches affect source code rather than by the order in which they are introduced into the codebase. This removes the temporal constraint among patches and enables the selection of patch sets based on their *semantic* dependencies. This patch dependency model is implemented in the Darcs version control system [153], which our software version-emulation architecture leverages to select consistent, conflict-free application versions for deployment.

6.1.3 Background

We provide a brief overview of work on honey-patching and understanding the threat model considered in this chapter.

Honey-patching

Prior work has observed that many vendor-released software security patches can be honeyed by replacing their attack-rejection responses with code that instead maintains and forks the attacker's connection to a confined, unpatched decoy [144, 154]. Such

honey-patching retains the most complex part of the vendor patch (the security check) and replaces the remediation code with some boilerplate forking code [155], making it easy to implement upon release of new security patches.

This embedded deception offers some important advantages over conventional honeypots. Most significantly, it observes attacks against the defender's genuine assets, not merely those directed at fake assets that offer no legitimate services. It can, therefore, capture data from sophisticated attackers who monitor network traffic to identify service-providing assets before launching attacks, who customize their attacks to the particular activities of targeted victims (differentiating genuine servers from dedicated honeypots), and who may have already successfully infiltrated the victim's network before their attacks become detected.

Threat Model

Attackers in our model submit malicious inputs intended to probe and exploit vulnerabilities on victim networked services. We assume most attackers rely upon a mix of vulnerabilities, only some of which are known to defenders. For example, a skilled attacker might first try to exploit known vulnerabilities, only escalating to more potent, defender-unknown vulnerabilities (e.g., 0-days) once they becomes confident that their activities are not being monitored. Attack payloads might be unique and therefore unknown to defenders. Such payloads might elude network-level monitors and are therefore best detected at the software level at the point of exploitation. We also assume that attackers might use one payload for reconnaissance but reserve another for the final attack. Misleading the attacker into launching the final attack is useful for defenders to discover the final attack payload, which can divulge attacker TTPs and goals not discernible from the reconnaissance payload alone.

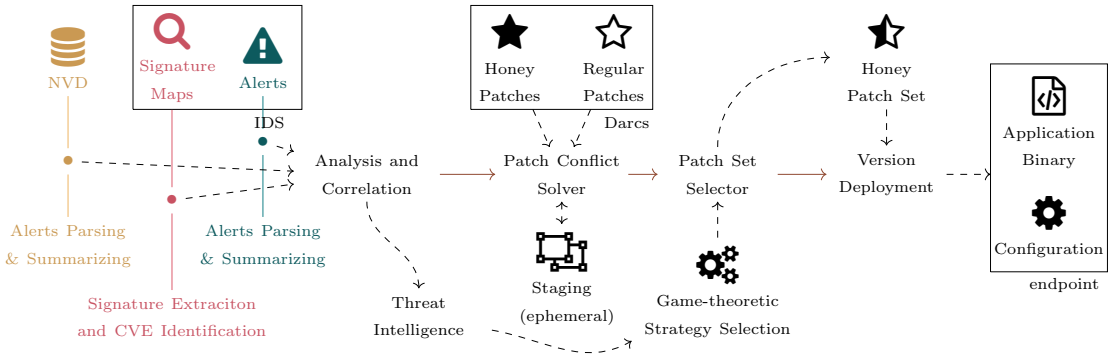


Figure 6.2: An overview of QUICKSAND.

While our general approach is potentially applicable to arbitrary networked software, in this work we focus on protecting services possessing strictly user-level privileges, and that must, therefore, leverage software bugs and kernel-supplied services to perform malicious actions, such as corrupting the file system or accessing other users' memory to access confidential data. QUICKSAND therefore instruments user-level applications with deceptive defensive code without modifying the OS or VM.

6.2 System Design

Figure 6.2 depicts QUICKSAND's architecture. The *patch conflict solver* generates conflict-free candidate patch sets for version emulation. An *analysis and correlation* component ingests and maintains vulnerability metadata from the National Vulnerability Database (NVD), parses intrusion alerts, and correlates them with intrusion signature metadata. The *patch set selector* module leverages a game-theoretic engine to select which version of the software should be deployed based on the aggregated data. The *version deployment* module then uses this information to synthesize and deploy a new version of the application, including the specification of the target modules and environment. This process executes repeatedly, and its trigger threshold can

be fixed, random, or dynamically adjusted (e.g., based on evidence and severity of intrusion alerts collected at the network perimeter).

Patch Theory

Darcs is a *change-based* version control system. In contrast to conventional history-based version control systems (e.g., Subversion, Git, CVS), which represent repository states as file trees, the state of a Darcs repository is defined by the set of patches it contains. This facilitates a *cherry-picking* operation—one that is not constrained by temporal dependencies among patches—that adds flexibility to our patch set selection model. Cherry-picking can be defined in terms of Darcs' underlying patch theory [156, 157], summarized as follows:

Definitions

The state of a repository is called a *context*. We write ${}^{o}A^{a}$ to denote that a repository moves from context o to context a via patch A. Patches are usually stored sequentially, and for any consecutive pair of patches, the final state of the first patch must be identical to the initial state of the second patch. A sequence of patches is written in left to right order, such as ${}^{o}A^{a}B^{b}C^{c}$ (or simply ABC if we omit contexts). Parallel patches share a common initial context and diverge to two different states ($A \vee B$).

Inversion

Every Darcs patch is invertible, affording the application of patches in either forwards or backwards directions to reach a particular context: $(AB)^{-1} = B^{-1}A^{-1}$. In par-

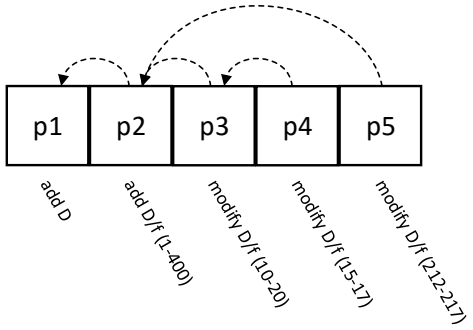


Figure 6.3: A repository state showing patch dependencies.

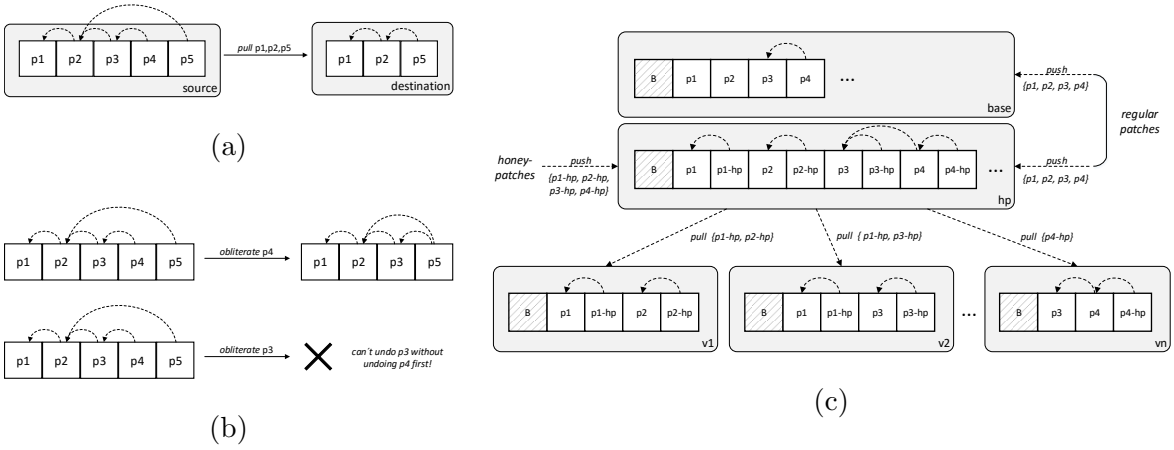


Figure 6.4: Operations illustrating (a) change-based patch cherry picking, (b) patch obliteration and consistency, and (c) patch management: patch set `B` denotes the set of patches making up the base source code of the software; patch dependencies pointing to it have been omitted.

ticular, AA^{-1} has no effect, and $(A^{-1})^{-1} = A$. Anti-parallel patches have different initial states yielding the same context ($A^{-1} \wedge B^{-1}$).

Commutation

The commutation of patches A and B is represented by $AB \leftrightarrow B'A'$, where A' and B' are intended to perform the same change as A and B. Intermediate states

may differ however: ${}^{o}A^{a}B^{b} \leftrightarrow {}^{o}B'^{x}A'^{b}$. A *merge* operation is defined as a pairwise computation, taking two parallel patches and converts them into a pair of sequential patches: $A \vee B \Longrightarrow AB' \leftrightarrow BA'$.

Cherry picking

Patch *cherry picking* refers to the ability to pull patches from a repository regardless of the order in which they were originally pushed into the repository. To illustrate, consider the repository state depicted in Figure 6.3. The repository consists of patches `p1`–`p5`, and the changes made by each patch are summarized underneath each patch. The dependencies between patches (denoted by dashed arrows) are computed by Darcs. Figure 6.4a illustrates cherry-picking for this particular example. Pulling patches `p1`, `p2`, and `p5` from the *source* onto the *destination* repository automatically adjusts the selected patches to fit the new context (without `p3` and `p4`). Darcs perform such adjustments using its patch manipulation algebra to allow users to reason about patches as sets, despite patches being stored as sequences internally.

Patch obliteration and consistency

Another advantage of patch commutativity is that patches can be *obliterated* (undone) without rolling back patches that historically succeed them. In the example above, patch `p4` can be removed from the repository without undoing `p5`, as illustrated in Figure 6.4b. To accomplish this, Darcs rearranges the sequence of patches by commuting `p4` with `p5`, and then removes `p4`. However, Darcs does not allow `p3` to be removed without first undoing `p4`; allowing this operation would constitute a patch dependency violation and render the state of the repository inconsistent.

Patch Management

Figure 6.4c illustrates our patch management strategy. Regular patches are *pushed* (stored) into Darcs repositories `base` and `hp`, and honey-patches are stored into repository `hp` only. We call `B` the set of patches that constitute the base version of the software (e.g., the initial commit, a specific tagged version of the application containing all patches up to the tag). Candidate versions selected by the patch selection module are stored as tags (e.g., `v1`–`vn`) by *pulling* specific patch sets from `hp`, which allows them to be easily retrieved for version deployment.

This patch management strategy leverages the underlying Darcs infrastructure, which automatically computes the transitive dependency relations for any given patch selection. For example, when pulling honey-patch `p4-hp`, Darcs correctly pulls patch set `B` and patches `p3`, `p4`, and `p4-hp`. This has the advantage of enabling a much simpler patch set generation algorithm (see §6.3).

Alert Analysis and Correlation

The alert analysis and correlation workflow pre-process vulnerability and environmental data to generate contextual information for patch set selection. First, intrusion alerts are parsed, and each alert class is annotated with descriptive statistics and target information. In the second step, the correlation module parses the intrusion detection system's signature map to extract the signature information for each alert object and cross-references it with the corresponding CVE identification derived from the reference field specified in the alert metadata. This step additionally filters intrusion alerts whose signatures target vulnerabilities that have not been identified as CVEs. The last step consults vFeed [158] to look up common vulnerability and exploit databases (e.g., CVSS, CWE, exploit-db) to aggregate threat intelligence metadata

```
{ cveID: CVE-2014-6277,
  targets: {('192.168.134.150', 80), ('192.168.134.139', 8080), ...},
  cvss: { 'vector': 'CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/...'},
  cwe: { id: 'cwe-78', term: 'OS Command Injection' },
  cpe: { 'cpe:/a:gnu:bash:2.02.1', 'cpe:/a:gnu:bash:2.01.1', ...},
  published: 2014-09-24T14:48:04.477-04:00,
  public_exploit: 'yes',
  count: 115 }
```

Listing 6.1: Alert object containing threat metadata

(e.g., vulnerability risk scores, exploit availability) and alert objects, which are used by the game-theoretic decision engine during the version selection process. Listing 6.1 shows an alert object containing threat metadata for CVE-2014-6277.

6.3 Moving Target Defense for Software Deception

To enable a truly dynamic system that makes it difficult for an adversary to fingerprint a deployed patch, QUICKSAND's patch selection process is built atop a moving target's defense-in-depth strategy combining cyber agility and honey-patching. In the context of software deception steering cyber maneuvers, we discuss the three components of this defense strategy described in Chapter 1: configuration set C, timing function T, and movement strategy M.

Configuration Set

The effectiveness of software deception steering depends on the selection of a set of code versions (with honey-patches) that can be deployed at any point in time. This requires each code version to be *conflict-free*.

Conflict-free Code Versions

Conflict in our system is defined by the following syntactic rule: if (honey-)patch A and (honey-)patch B prescribe different contents for the same line of code, then A and B cannot coexist automatically in the same code version. A code version can be viewed as an element in the power-set of patches, i.e., $v \in \wp(\Pi)$. Thus, pruning this power set based on the pair-wise definition of conflict between patches results in the configuration set C for the MTD.

Algorithm 4 details (in pseudocode) the algorithm for generating conflict-free code versions (or patch-sets) given the inputs Π representing the set of available security patches, the base repository B containing regular patches, and the repository HP of honey-patches. Lines 4–5 initializes the conflict-set cs to be empty, and a temporary repository Δ as a copy of B. The algorithm then populates cs with all conflicting patch pairs $\in \Pi \times \Pi$ (or Π^2) by checking the result of merging the corresponding honey-patch pair from HP into Δ and then resetting Δ between each merge operation (lines 6–11). Line 12 removes the temporary repository. Finally, in Line 13 the set of conflict-free patch sets is generated by pruning out all code versions from $\wp(\Pi)$ that contain conflict-pairs. While complex pruning rules (such as allowing honey-patches only to releases officially reported in the Common Platform Enumerations database) can be specified, it reduces the cardinality of the configuration settings and therefore the available options for the cyber maneuver.

Timing Function

QUICKSAND uses an event-based timing function T. In this setting, when alerts are triggered by our system, we use it to compute the existence of a particular attacker type (discussed in the next section) and adapt our current deception strategy given

Algorithm 4 Conflict-free patch set generation algorithm

procedure
GIVEN Π: PATCH SET, B: BASE REPOSITORY, HP: HONEY-PATCH REPOSITORY,
OUTPUT(set of conflict-free patch sets)
    $cs \leftarrow \emptyset$
    $\Delta \leftarrow B$
    for $(p_1, p_2) \in \Pi^2$ **do**
        if $\neg$pull($\{p_1, p_2\}, HP, \Delta$) **then**
            $cs \leftarrow cs \cup \{(p_1, p_2)\}$
        end if
        obliterate('[.*]-hp$', Δ)
    end for
    remove(Δ)
    return $\{S \in \wp(\Pi) \mid S^2 \cap cs = \emptyset\}$
end procedure

this knowledge. In scenarios where alerts are ubiquitous, we consider a hybrid T that uses aggregate alert information over a time-period to modify the movement strategy.

6.3.1 Game-theoretic Movement Function

Given the set of conflict-free code versions, the system must decide which version is to be deployed at the time of switching. To this end, we first consider a *game-theoretic modeling* of the interaction between an adversary and QUICKSAND. Then, we define an optimal deception selection strategy and describe methods to compute it.

In real-world settings, defenders often target adversaries having a particular set of characteristics. Curating the patch set selection strategy to the nuances of this adversarial profile thus results in more effective countermeasures. For example, it is ineffective to honey-patch only older, nearly obsolete vulnerabilities to gather threat intelligence about expert adversaries armed with the newest exploits. To address this concern, we model the problem as a two-player Bayesian Stackelberg Game (BSG) inspired by prior MTD web defense models [20].

Our BSG can be defined as an n-tuple $\langle \mathcal{D}, \mathcal{A}, A^{\mathcal{D}}, A^{\mathcal{A}}, R^{\mathcal{D}}, R^{\mathcal{A}}, P \rangle$ where,

- $\mathcal{D}$ denotes the defender,
- $\mathcal{A} = \{\mathcal{A}_1, \ldots, \mathcal{A}_\theta\}$ denotes the θ types of attacker,
- $A^{\mathcal{D}}$ and $A^{\mathcal{A}} = \{A_1^{\mathcal{A}}, \ldots, A_\theta^{\mathcal{A}}\}$ denote the action-sets A of the players,
- $R^{\mathcal{D}} = \{R_1^{\mathcal{D}}, \ldots, R_\theta^{\mathcal{D}}\}$ and $R^{\mathcal{A}} = \{R_1^{\mathcal{A}}, \ldots, R_\theta^{\mathcal{A}}\}$ denote their utilities (with the subscripts representing the utility of the players corresponding to the adversary's type), and
- $P = \{P_1, \ldots, P_\theta\}$ denotes a probability distribution that represents the likelihood of facing each attacker type.

Note that this is similar to the BSG model defined in Chapter 3 but without switching costs. Thus, similar to [61, 20], our goal is to infer a robust deception strategy, in expectation, against all attacker types. Next, we discuss how each of these model parameters is obtained and use real-world examples to elucidate the descriptions.

Players ($\mathcal{D}, \mathcal{A}$). The defender $\mathcal{D}$ represents the administrator who sets up the system, inspects reported alerts, and chooses to deploy a particular deception measure.

The attacker $\mathcal{A}$ has three types according to skill set level—script kiddie ($\mathcal{A}_1$), early adopter ($\mathcal{A}_2$), and APT attacker ($\mathcal{A}_3$). As the names suggest, $\mathcal{A}_3$ is an expert who spends time in identifying vulnerabilities against a system, whereas $\mathcal{A}_1$ only uses attacks that have a publicly available implementation of exploits, and $\mathcal{A}_2$ is biased towards exploits that are trending. A more formal distinction follows in our discussion of player action sets.

Actions (A). The defender's actions $A^{\mathcal{D}} = C = \{v_1, \ldots, v_n\}$ consist of the n feasible candidate patch-sets found using Algorithm 4. A defender can choose any of these actions (also referred to as a pure strategy) at any point in time. Given the patch sets used by the defender, we compile a list of known exploits E that can be used by an attacker. We enumerate the exploits against our system in Table 6.1 and consider subsets of E, using the `published` and the `public_exploit` fields in the metadata object of the exploit (see Listing 6.1), to describe the exploits available to each attacker type.

The script-kiddies ($\mathcal{A}_1$) attack set consists of CVEs that have a known public exploit available. The early-adopters ($\mathcal{A}_2$) attack set comprises vulnerabilities that have a public exploit available for which the published date is less than $t = 5$ years. Thus, $\mathcal{A}_2$'s attack set includes the last two CVEs in the list shown in Table 6.1 (i.e. $|E_2| = 4$). The APT attacker ($\mathcal{A}_3$) can write exploits for any of the existing CVEs in the list, and has all the attack actions available (i.e. $|E_3| = |E| = 12$).

Utilities/Rewards (R). To design the utility for the players, we primarily consider metrics that are a part of the Common Vulnerability Scoring System (CVSSv3) [159]. We first define a generic reward structure and discuss how it can capture the various

Vulnerability	Description	Software	CVSS		
			Impact (I)	Exploitability (E)	Overall (O)
CVE-2014-0160	Information leak	Openssl	5.4 CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N/E:H/RL:O/RC:C/CR:H/IR:X/AR:X/MAV:N/MAC:L/MPR:N/MUI:N/MS:U/MC:H/MI:N/MA:N	3.7	8.9
CVE-2012-1823	System remote hijack	PHP	3.4 CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:L/E:H/RL:O/RC:C/CR:M/IR:M/AR:M/MAV:N/MAC:L/MPR:N/MUI:N/MS:U/MC:L/MI:L/MA:L	3.7	7.0
CVE-2011-3368	Port scanning	Apache	1.4 CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N/E:P/RL:O/RC:C/CR:M/IR:X/AR:X/MAV:N/MAC:L/MPR:N/MUI:N/MS:U/MC:L/MI:N/MA:N	3.5	4.8
CVE-2014-6271	System hijack	Bash	6.1 CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H/E:H/RL:O/RC:C/CR:H/IR:H/AR:H/MAV:N/MAC:L/MPR:N/MUI:N/MS:C/MC:H/MI:H/MA:H	3.7	9.5
CVE-2014-6277	System hijack	Bash	6.1 CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H/E:H/RL:O/RC:C/CR:H/IR:H/AR:H/MAV:N/MAC:L/MPR:N/MUI:N/MS:C/MC:H/MI:H/MA:H	3.7	9.5
CVE-2014-0224	Session hijack and information leak	Openssl	5.9 CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:N/E:F/RL:O/RC:C/CR:H/IR:H/AR:X/MAV:N/MAC:H/MPR:N/MUI:N/MS:X/MC:H/MI:H/MA:X	2.1	7.5
CVE-2010-0740	DoS via NULL pointer dereference	Openssl	2.1 CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:L/E:P/RL:O/RC:C/CR:X/IR:X/AR:H/MAV:N/MAC:L/MPR:N/MUI:N/MS:U/MC:N/MI:N/MA:L	3.5	5.5
CVE-2010-1452	DoS via request that lacks a path	Apache	1.4 CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:L/E:P/RL:O/RC:C/CR:X/IR:X/AR:M/MAV:N/MAC:L/MPR:N/MUI:N/MS:U/MC:N/MI:N/MA:X	3.5	4.8
CVE-2016-6515	DoS via request that lacks a path	OpenSSH	5.4 CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H/E:H/RL:O/RC:C/CR:X/IR:X/AR:H/MAV:N/MAC:L/MPR:N/MUI:N/MS:U/MC:N/MI:N/MA:H	3.7	8.9
CVE-2016-7054	DoS via heap buffer overflow	Openssl	5.4 CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H/E:H/RL:O/RC:C/CR:X/IR:X/AR:H/MAV:N/MAC:L/MPR:N/MUI:N/MS:X/MC:X/MI:X/MA:H	3.7	8.9
CVE-2017-5941	System hijack	Node.js	4.0 CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H/E:H/RL:O/RC:C/CR:L/IR:L/AR:L/MAV:N/MAC:L/MPR:N/MUI:N/MS:U/MC:H/MI:H/MA:H	3.7	7.6
CVE-2017-7494	System hijack	Samba	5.9 CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H/E:H/RL:O/RC:C/CR:H/IR:H/AR:L/MAV:N/MAC:L/MPR:N/MUI:N/MS:U/MC:H/MI:H/MA:H	3.7	9.4

Table 6.1: Summary of (honey-)patchable vulnerabilities with the corresponding CVSS scores.

aspects of cyber deception. Then, we highlight how we can obtain numeric values that represent the utility of the players.

The utility structure for a player, given that defender $\mathcal{D}$ deploys code version v and attacker $\mathcal{A}_i$ executes an exploit $e \in E_i$, is as follows:

$$R_i^{\mathcal{D}}(v,e) = \begin{cases} +r_i^{\mathcal{D}}(v,e) - c(v) & \text{if } e\text{-hp} \subset v \\ -I^{\mathcal{D}}(e) - c(v) & \text{otherwise} \end{cases}$$

$$R_i^{\mathcal{A}}(v,e) = \begin{cases} -r_i^{\mathcal{A}}(v,e) - c(e) & \text{if } e\text{-hp} \subset v \\ +I^{\mathcal{A}}(e) - c(e) & \text{otherwise} \end{cases}$$

In the first case, where the code version deployed has a honey-patch for exploit e that the attacker decides to exploit, the reward for the defender has two components. First, $\mathcal{D}$ gets a positive reward of $r_i^{\mathcal{D}}(v,e)$ because the attacker $\mathcal{A}_i$ was trapped using the honey-patch. This value is specific to the exploit being honey-patched as it needs to account for its intelligence-gathering worth (e.g., IPs used by the attacker) combined with the actionable protective measures that can be taken (e.g., add such IPs to the firewall deny-list). In this regard, given the attacks in our system, we consider the following ordering: $r_i^{\mathcal{D}}(v, \text{DoS}) \leq r_i^{\mathcal{D}}(v, \text{Port Scanning}) < r_i^{\mathcal{D}}(v, \text{Info Leakage}) < r_i^{\mathcal{D}}(v, \text{System Hijacking})$.

To simplify our analysis, we omit some of the context-specific information about the operating environment (e.g., the value of targeted assets, mission-critical requirements) and attacks (e.g., targeted scanning, distributed *vs.* targeted DoS); this allows us to disregard the equality conditions and design uniformly spaced rewards in the range $[0, \max_e I^{\mathcal{D}}(e)]$. Second, the value $c(v)$ represents the cost of deploying a particular honey-patch on the Quality of Service (QoS) metrics on a system. We assume that all conflict-free patch sets v have the same cost and ignore the term altogether (as these values cannot incentivize $\mathcal{D}$ to pick a particular code version based on QoS metrics).

Attacker $\mathcal{A}_i$, when trapped by a honey-patch, incurs the cost of crafting and executing the exploit $c(e)$. This cost is dependent on the complexity of an attack (represented by its exploitability score) and the temporal metrics (what kind of an exploit or patch is available and how reliable the source is). Thus, we multiply the exploitability score (ES) of CVSS with the temporal metrics to obtain $c(e)$, similar to the way temporal scores are obtained using the base score (BS). The other negative reward $r_i^{\mathcal{A}}(v, e)$ captures $\mathcal{D}$ gaining knowledge of an attacker's TTPs. In our model, we assume that the attacker is unaware of which vulnerabilities are honey-patched, and this $r_i^{\mathcal{A}}(v, e) = 0 \ \forall i \in \{1, \ldots, \theta\}$. One may choose to assign different scores to different players. For example, this can be used to reflect the fact that an APT attacker ($\mathcal{A}_3$) is better equipped to detect the deception.

In the second case, when v does not contain a honey-patch for e, the attacker can either gain reconnaissance if a regular patch is deployed by leveraging the attack failure information or cause full impact without getting caught if no regular patches are available (for relatively new CVEs). For the latter case, $\mathcal{D}$ receives a negative utility against $\mathcal{A}_i$ with magnitude equal to the impact score, while $\mathcal{A}_i$ receives a positive utility with magnitude equal to the overall score. The overall score trades off the impact of the attack alongside the complexity of constructing and executing it. Table 6.1 shows the CVSSv3 metrics corresponding to the individual exploits of the formulated game, leveraged to calculate the utilities. For the former case, $\mathcal{D}$'s loss is a fraction of the impact score for giving out attacker info, whereas $\mathcal{A}_i$ considers its effort cost and the utility of gathering the information about patches.

Attacker Type Probabilities (P). We start with an initial probability distribution over attacker types (denoted as $\langle \Pr(\mathcal{A}_1), \ldots, \Pr(\mathcal{A}_\theta) \rangle$) provided by analysis of historical data by security experts and historical data from similar systems. Based

on an alert a raised by the analysis and correlation module, we update the attacker type probabilities P_i using the Bayesian rule over $\Pr(\mathcal{A}_i)$:

$$\begin{aligned} P_i &= \Pr(\mathcal{A}_i|a) \\ &= \alpha \Pr(\mathcal{A}_i) \cdot \Pr(a|\mathcal{A}_i) \\ &= \alpha \Pr(\mathcal{A}_i) \cdot \sum_e Pr(a|e) \cdot \Pr(e|\mathcal{A}_i) \\ &= \alpha \Pr(\mathcal{A}_i) \cdot \sum_e Pr(a|e) \cdot \mathbb{I}(e \in \mathcal{A}_i) \end{aligned}$$

where α represents the normalization factor and $\mathbb{I}$ represents an indicator function that equals 1 if the condition is met and 0 otherwise. Ideally, the value $\Pr(e|\mathcal{A}_i)$ should converge to the strategy of an attacker type if they were to behave rationally. However, an alert may be observed for any of the available exploits, even when it is a sub-optimal choice for a rational adversary.

Hence, we account for this irrationality by using an indicator function. Further, if an alert can be generated by multiple exploit actions e, an attacker type with a larger number of such exploits should be assigned higher probability. Lastly, while some alert systems, such as anomaly detection systems based on machine learning, may detect certain exploits imperfectly, we limit ourselves to deterministic detection mechanisms $\Pr(a|e) \in \{0, 1\}$.

As an example, consider the use of a CVE from 2014 that is distinctive in observing a particular system alert. Given $\mathcal{A}_2$ cannot perform this attack action, P_2 becomes zero. This probability is distributed between P_3 and P_1, as $\mathcal{A}_1$ or $\mathcal{A}_3$ may have generated this alert. Thus, the initial distribution $\langle 0.4, 0.4, 0.2 \rangle$ becomes $\langle 0.67, 0, 0.33 \rangle$.

6.3.2 Strategy Computation

A strong threat model must account for an adversary capable of performing reconnaissance on the targer. As discussed in the previous chapter, this boils down to the use of *Stackelberg Equilibrium*, which encodes the assumption that the defender acts as a leader while the attacker, who is aware of the defender's deployment strategy, assumes the role of a follower [20, 35]. In this setting, the defender plays a mixed strategy (i.e., a probabilistic strategy over his actions) making it impossible for the attacker to fingerprint the deception strategy in place at any given point in time. In this Bayesian Stackelberg Game setting, we can calculate the optimal movement strategy ($\vec{x}$) for the defender by maximizing $\mathcal{D}$'s expected utility, as follows,

$$\sum_{i} \sum_{v} \sum_{e} P_i \; x_v \; q_e^i \; R_i^{\mathcal{D}}(v, e) \tag{6.1}$$

where each attacker $\mathcal{A}_i$'s strategy $\vec{q}^{\,i}$ is calculated with the knowledge of the defender's strategy $\vec{x}$ by maximizing $\sum_e \sum_v x_v \; q_e^i \; U_i^{\mathcal{A}}$, subjected to constraints that $\vec{q}^{\,i}$ represents a probability distribution. Both optimizations—maximizing the defender's expected utility given the attacker maximizing their utility—can be folded into a single optimization problem [103]. As per our discussion in Section 3.3, we use the Mixed-Integer Quadratic Program (MIQP) formulation instead of the Mixed Integer Linear Program (MILP) for (1) efficiency reasons and (2) calculating the strategy of version emulation in QUICKSAND. At the start of each period, we use this mixed strategy to select a particular code version at random and proceed with its deployment.

Version Deployment

Upon completion of patch selection, QUICKSAND deploys a new version of the application into the target environment. Figure 6.2 outlines the steps taken to deploy an

```
[Apache-1]
app = apache
base_repo = ../data/base
hp_repo = ../data/hp
deploy_repo = ../data/deploy
configure_command = make
install_command = make install
patches = CVE-2014-0160:CVE-2014-6271: \
        CVE-2014-6277:CVE-2014-7169: ...

[Apache-2]
app = apache
...

[OpenSSL]
app = openssl
...
```

Listing 6.2: QUICKSAND example configuration file

application. The first step consists of creating a *working* repository for the application, by first pulling all patches from `base` into `target`, and then pulling only the selected honey-patch subset into `target`. This yields a working repository state that is *tagged* with the selected application version. The final step consists of building the target application from sources, using a user-supplied configuration as supplemental input. The configuration parameters are specified per application, as shown in the configuration file illustrated in Listing 6.2, to set up the built environment and release the new application version.

6.4 Implementation

We developed an implementation of QUICKSAND for the 64-bit version of Linux. The implementation consists of four Python components: the *repository handler* module consists of about 150 lines of code and wraps Darcs CLI [157] to offer an API to access the version control system. The *analyzer* component consists of 90 lines of code, and leverages *py-idstools* [160] to parse IDS signature maps and events sourced in unified2 format (a serialized binary stream format specification for IDS events), and *vFeed* [158] to fetch and aggregate threat metadata to alert objects. The *patch selector* module consists of an additional 140 lines of code, and the *version deployment* module adds about 80 lines of code to the system. Our implementation depends on a deployment environment that has been pre-configured with a honey-patching framework, along with its process sandboxing and monitoring facilities [144].

6.4.1 Conflict-free Patch Sets

Table 6.2 summarizes the conflict-free patch sets used as inputs to our game-theoretic decision process. These serve as candidate *versions* for patch selection, and encode information about affected software, such as application version compatibility. Each patch set implicitly encodes the availability of regular (e.g., CVE-2017-7494) and honey (e.g., CVE-2017-7494-hp) patch selections. Moreover, our patch repository maintains patch metadata that is used to filter *unpatchable* patch sets—patch selections for which a version deployment is infeasible due to patch compatibility and operation requirements.

Table 6.2: Summary of conflict-free patch versions (-hp implied).

#	Patch Set	Affected Software (CPE)
1	CVE-2014-0160	openssl:1.0.1f (≤)
2	CVE-2014-6271, CVE-2014-6277	bash:[4.3, 3.2.48, 2.0.5, 1.14.7] (≤)
3	CVE-2014-0160, CVE-2014-6271, CVE-2014-6277	openssl:1.0.1f (≤), bash:[4.3, 3.2.48, 2.0.5, 1.14.7] (≤)
5	CVE-2011-3368	http_server:[2.2.21,2.0.64,1.3.68] (≤)
6	CVE-2010-1452, CVE-2011-3368	http_server:2.2.15 (≤)
7	CVE-2012-1823	php:[5.4.1, 5.3.10] (≤)
8	CVE-2016-6515	openssh:7.2 (≤)
9	CVE-2014-0224	openssl:[1.0.1f,1.0.0l,0.9.8y] (≤)
10	CVE-2014-0160, CVE-2014-0224	openssl:1.0.1f (≤)
11	CVE-2010-0740	openssl:0.9.8m (≤)
12	CVE-2010-0740, CVE-2014-0224	openssl:0.9.8m (≤)
13	CVE-2016-6515, CVE-2014-0224	openssh:7.2 (≤), openssl:1.0.1f (≤)
14	CVE-2016-6515, CVE-2010-0740	openssh:7.2 (≤), openssl:0.9.8m (≤)
15	CVE-2016-6515, CVE-2014-0224, CVE-2010-0740	openssh:7.2 (≤), openssl:0.9.8m (≤)
16	CVE-2017-5941	node-serialize:0.0.4 (≤)
17	CVE-2017-7494	samba:[4.1.23, 4.0.26,3.6.25,3.5.22] (≤)

6.4.2 Simulation Results

Table 6.3 highlights simulation results obtained using three different movement strategies: *static deception*, *uniform random strategy* (URS), and *Bayesian Stackelberg Game* (BSG). For simulation, we assume that the four latest vulnerabilities cannot be patched due to lack of officially available patches; the defender faces maximum impact for these and smaller impact for other vulnerabilities that are regularly patched due

Strategy	Expected Utility ↑	# Patch-sets used ↓
Static	-5.87	**1**
URS	-1.66 ± 0.81	17
BSG	$\mathbf{-1.26 \pm 0.89}$	11

Table 6.3: Benefits of different patch selection strategies.

to leakage of reconnaissance information. The reward values for the defender shown in Table 6.3 are plotted across 12 different runs. In each run, we consider a distinct exploit is detected and update the beliefs over the attacker types accordingly.

In comparison to a static strategy that deploys the most profitable honey-patch set, software deception steering, regardless of employed movement strategy, increases system administrators' expected utility. Although, in our game setup, the use of uniform random strategy (URS) (i.e., selecting either of the 17 versions with equal probability at deployment time) is sub-optimal when compared to the Bayesian Stackelberg Equilibrium strategy. Further, the game-theoretic strategy identifies 6 code-versions devoid of security benefits, significantly reducing the defender's overhead of maintaining all 17 patch-sets. While the patch-sets #12 and #14 are pruned-out as #15 is a strict-subset, we also notice path-set #13 has a non-zero deployment probability. The patch-sets $\{1, 8, 9, 10, 12, 14\}$ are assigned zero-probability of deployment. Among the patch-sets that have non-zero probabilities of deployment (on average across the 12 runs), 5 of them have the highest deployment probability of 14.4% and the patch-set #7 has the lowest deployment probability of 1.1%.

While our game parameters are based on simulation results over rewards obtained from security databases, human-subject case studies must be conducted to understand the true benefits of this model. Thus, validation of QUICKSAND under

these three movement strategies in empirical attack-defense exercises is an important future direction.

6.5 Concluding Remarks

Our approach enhances vulnerability patching in software security with a moving target defense technique that makes applications less predictable and more robust against attackers' anti-deception efforts. Toward this end, we designed and implemented an adaptive, software version-emulation architecture in which the set of honey-patched vulnerabilities in a target application is dynamically re-selected to increase the likelihood of deceiving and entrapping attackers. Leveraging our game-theoretic analysis that automates and optimizes (honey-)patch management, our framework infers effective movement strategies based on contextual threat metadata and attacker model.

Chapter 7

MOVING TARGET DEFENSE FOR DEEP NEURAL NETWORKS IN MACHINE LEARNING SECURITY

Table of Contents ◇ 1 ◇ 2 ◇ 3 ◇ 4 ◇ 5 ◇ 6 ◇ 7 ◇ 8 ◇ 9

C	Attack and Exploration Surface Shifting in Machine Learning
t	On-event switching
M	Stackelberg Strategy of a Bayesian Normal-form Game

The recent popularity of machine learning in general and deep neural networks, in particular, have made it an integral part of many modern software systems. While its inclusion has improved the state-of-the-art in recognizing patterns and anomalies across multiple data modalities, it has also introduced a new (and as we shall see, an easy) attack surface for the adversary to exploit [6]. Defenses against adversarial examples are designed to be effective against a certain class of attacks by either training the classifier with perturbed images generated by these attacks or making it hard for these attacks to modify some property of the neural network. Unfortunately, this has evolved into a cat-and-mouse game and often, a state-of-the-art defense mechanism is proven to be inadequate against a new class of attacks almost as soon as it is published. Some recent works try to formulate the attack scenario as a min-max adversarial game where the defender tries to minimize the loss while the attacker tries to maximize it. They show the use of Projected Gradient Descent (PGD) for

solving the inner (max) optimization can result in attacks that are extremely effective in crippling the classification system and, at the same time, capture the characteristics of many of state-of-the-art attacks [161]. They claim that robust training methods that enforce classification to the same class when images are ϵ distance away from any image in the training set results in high classification accuracy against adversarial examples. Unfortunately, this has the side effect of bringing down the classification accuracy on non-perturbed examples.

In this chapter, we take a novel view and design a meta-defense that can function both as (1) the first line of defense against new attacks and (2) defense-in-depth solution used in conjunction with any existing defense mechanism to boost the security gains provided. To this end, we design Moving Target Defense (MTD) that randomly selects a network from an ensemble of networks when classifying an input image (randomization at test time), for boosting the robustness against adversarial attacks (see Fig. 7.1). Precisely, our contributions can be highlighted as follows.

- MTDeep – an MTD framework for an ensemble of DNNs, which can be used as a meta-level defense-in-depth mechanism, to bootstrap any existing defense mechanism and increase the robustness of the classification system to different classes of adversarial attacks.

- A Bayesian Stackelberg Game formulation with two players– MTDeep and its users. We show that the Strong Stackelberg Equilibrium of this game results in the optimal strategy for test-time selection of constituent networks in the ensemble. The strategy maximizes the classification accuracy on regular as well as adversarially modified inputs.

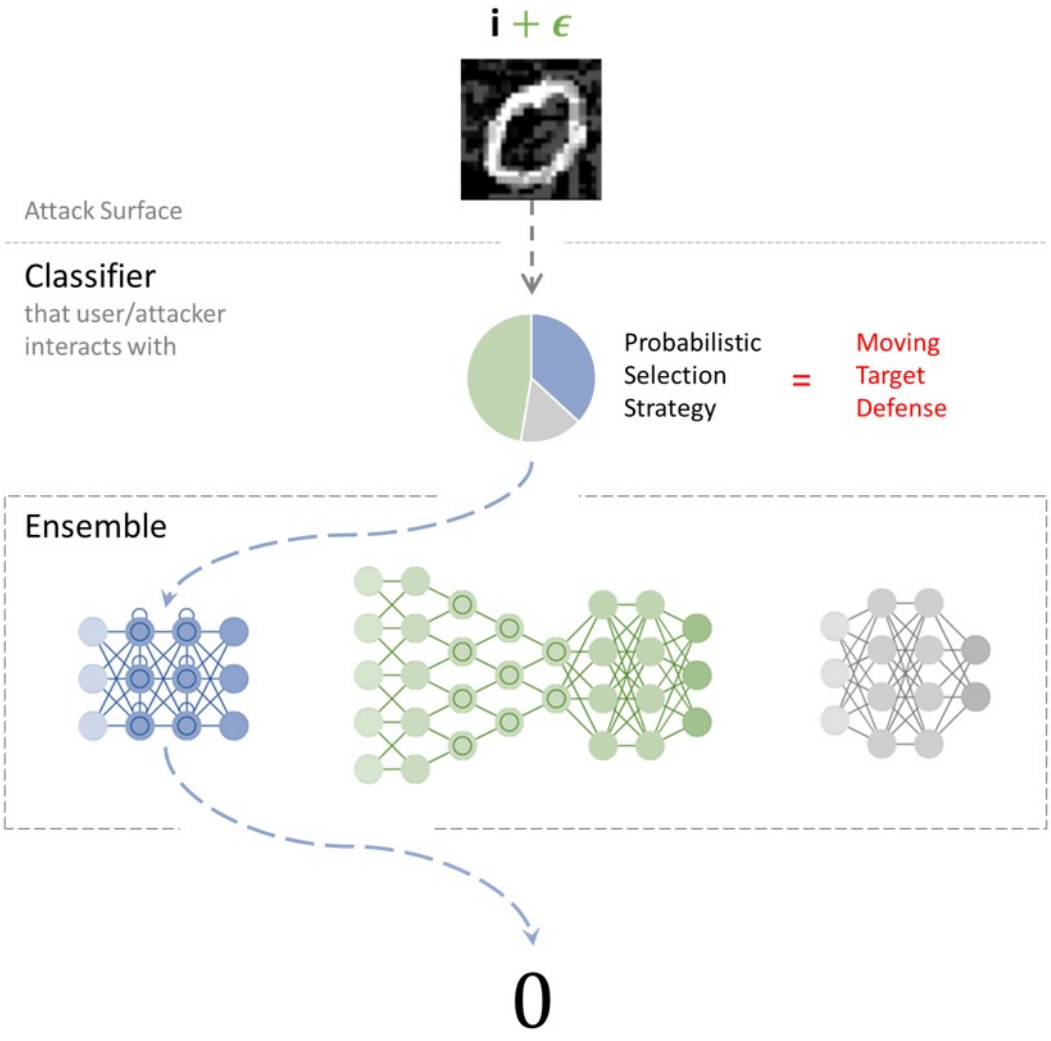


Figure 7.1: MTDeep– Moving Target Defense for Deep Neural Networks. In this example, an attacker fails at making an image classification system misclassify adversarially perturbed MNIST examples. The adversary chooses to perturb the image of a '0' with an attack that works for the (green) Hierarchical Recurrent Neural Network (HRNN) in the ensemble, but upon feeding it as input to MTDeep, the system rolls dice to picks the (blue) Multi-Layer Perceptron (MLP) that correctly classifies the input image to zero because the MLP was immune to the adversarial perturbation crafted by the adversary for HRNN.

- Empirical evaluation to show that MTDeep can be used as (1) a standalone defense mechanism to increase the accuracy on adversarial samples by $\approx 24\%$ for MNIST, $\approx 22\%$ for Fashion MNIST and $\approx 21\%$ for ImageNET data-sets against a variety of well-known attacks and (2) in conjunction with existing defense mechanisms like Ensemble Adversarial Training, MTDeep increases the robustness of a classification system (by $\approx 50\%$ for MNIST). We also show that black-box attacks (c.f. related work) on a distilled network are ineffective (in comparison to white-box attacks) against the MTDeep system.

- Analysis of an ensemble of DNNs for MNIST data that elucidates how much of a security gain MTDeep can provide. As a part of that analysis, we define the concept of *differential immunity*, which is (1) the first attempt at defining a robustness measure for an ensemble against attacks and (2) a quantitative metric to capture the notion of attack transferability.

7.1 Background

In this section, we introduce to works on crafting adversarial inputs against deep neural networks (at test-time) and defenses developed against them.

Attacks and Defenses for Deep Neural Networks

Gradient-based perturbations

Recent literature has shown multiple ways of crafting adversarial samples for a DNN [162, 163, 6, 161]. In these works, either (i) the input features that have high influence on the DNN's loss (measured via the partial derivatives) are modified by a small amount to maximize the DNN's loss function and therefore make the classifier

misclassify them, or (ii) the geometric space around a point is examined to find the closest class-separation boundary and generate perturbation vectors that push the modified image to the other side of this boundary. Similar to a chosen-ciphertext attack, these attacks assume that the test image which is to be modified is available beforehand. Furthermore, they assume the availability of complete knowledge about the classification network.

Black-box attacks

The threat model of black-box attacks considers the attacker does not have access to the network parameters but can interact with it. Often, attackers can train a small substitute model trained using selected inputs and the output labels provided by the network being attacked [164], similar to chosen-plaintext attacks. Surprisingly, attacks on this substitute model generalize well to the actual network [6]. Recent work on zeroth-order optimization has shown it is possible to create black-box attacks without the need for substitute models [165].

Defenses

Defense techniques against the two types of attacks described above commonly involve (1) generating adversarial perturbed training images using one (or all) of the attack methods described and then (2) using the generated images along with the correct labels to fine-tune the parameters of the DNN during training. This helps the DNN to correct its bias in some of the unexplored areas of the high dimensional space, reducing the effectiveness of the adversarial perturbations. Ensemble adversarial training [166] and stability training [167] are two improvements on this defense

technique. [1] Besides these, researchers have developed defense mechanisms like gradient masking [168], defensive distillation [169] and dimensionality reduction & 'anti-whitening' [170]. Some of these defense mechanisms, similar to trends in cyber-security, have been rendered ineffective due to the discovery of stronger attacks (eg. [171]). We do not consider these methods further since *our proposed framework can be used in conjunction with any of these to improve their security guarantees.* Later we will piggyback our proposed MTD with the state-of-the-art defense mechanism that uses adversarial training with the Projected Gradient Descent (PGD) attack to improve it further.

Universal perturbations

This DNN-specific perturbation when added to any input image, makes a DNN misclassify it [172]. This attack is based on the DeepFool attack [162] and although it is often time-consuming to generate, only one "universal" perturbation per network needs be computed. Moreover, the authors show that adversarial training is ineffective in increasing robustness against these attacks. Also, other state-of-the-art defense mechanisms ([166, 161]) have not shown that they can mitigate this attack. The newer class of such DNN-specific attacks such as Adversarial Patches [173] and BadNets [174] relax the constraint that the perturbation is imperceptible to a human. We show that randomly switching between networks of an ensemble to classify an input image, as MTDeep does, can prove to be an effective defense against these attacks because these attacks are network-specific and often have low transferability.

[1] Ensemble Adversarial Training uses an ensemble to generate adversarially perturbed examples for all the constituent networks and uses them to strengthen a single network [166]. Unlike us, it does not use the ensemble at classification time.

There has been some effort in trying to protect machine learning systems against attacks using randomization techniques at test time [175]. Unfortunately, these are not general enough to be used for DNNs. Furthermore, these mechanisms try to prevent misclassification rates under attack and end-up affecting the classification accuracy on non-adversarial or legitimate test inputs.

Existing Randomization or Ensemble-based Defense Attempts

There has been some effort in trying to protect machine learning systems against attacks using randomization techniques [175]. Unfortunately, these are not general enough to be used for DNNs. Furthermore, these mechanisms try to prevent misclassification rates under attack and end up affecting the classification accuracy on non-adversarial or legitimate test inputs. Lastly, our approach is further supported by previous research works that show the introduction of randomized switching makes it harder for an attacker to reverse engineer a classification system with precision [176], which is how most white-box attacks are constructed.

The general concept of DNN-based ensembles simply tries to increase classification accuracy for legitimate users but provides no protection against adversarial examples [177]. Ensemble Adversarial Training (EAT), as discussed above, trains a single classifier based on the adversarial inputs generated for all constituent networks in the ensemble [166]. Thus, EAT is a single-classifier system opposed to being an ensemble. Deviating from this, in [178], researchers propose an ensemble-based method to detect adversarial samples for the MINST dataset. Note that, we can extend this idea to design a voting based ensemble of DNNs to increase the robustness of the overall classification system. Unfortunately, such voting based mechanisms for ensembles can be viewed as simply adding an extra pooling layer whose weights are

equal to the weightage given to the votes of the constituent networks. For example, consider we have an ensemble with two constituent DNNs (D_1 and D_2) for a three-class classification problem (c_1, c_2, c_3). Let us say that the classification result of D_1 is given weight w_1 and the classification result of D_2 is given weight w_2. One can view this setup as an equivalent DNN D' whose last layer has a connection from c_i classes of each constituent D_j to the output class c_i of D'. At this point, all attacks on a DNNs described above, generalize to these voting-based ensembles. Researchers have also shown that an ensemble of vulnerable DNNs cannot result in a classifier robust to attacks [179]. In contrast, MTDeep builds in an implicit mechanism based on randomization at prediction time, making it difficult for an adversary to fool the classification system.

A particularly exciting development of late [180] has demonstrated how an exponentially large ensemble of models can be generated from a single model to provide robustness against the transferability of attacks. This perfectly complements our approach which leverages the concept of "differential immunity" of an ensemble to provide the first line of defense against adversarial images. Lastly, note that earlier works have proposed robustness measures only for a single network (either attack-specific [181] or attack-independent [182]). We introduce the concept of attack-specific robustness measures for an ensemble.

Although prior research has shown that the effectiveness of attacks can sometimes transfer across networks [6], we show that there is still enough residual disagreement among networks that can be leveraged to design an add-on defense-in-depth mechanism by using MTD. In fact, recent work has demonstrated that it is possible to train models with limited adversarial attack transferability [180], making our meta-level defense approach particularly attractive.

As stated in the previous chapters, a Moving Target Defense (MTD) seeks to, at every time instance based on a timing function t, randomly deploy a particular configuration from a set of system configurations C. The attacker, who has a set of attacks, tries to cripple the MTD system. As the attacker does not know which system was specifically selected at a particular time instant, its attacks become less effective (Figure 7.1). While randomization is essential for the effectiveness of MTD, in the context of classification systems, it might end-up reducing the accuracy of the overall system in classifying legitimate inputs drawn from the test-set because the DNN has the highest accuracy may not always be chosen for classification. Thus, in order to retain good classification accuracy and guarantee high security, we model the interaction between MTDeep and the users as a Bayesian normal-form Game. In this section, we describe the various elements of the game-theoretic model.

The Defender ($\mathcal{D}$)

The defender (MTDeep) provides a service for the classification of images. The configuration space for MTDeep is the DNNs in the ensemble that is trained on the particular image classification task. Let C represent the set of neural network configurations, which constitutes the defender's pure-strategy set, i.e. $M^{\mathcal{D}} = C$. In the ensemble we design for our experiments on MNIST and Fashion-MNIST datasets, we have three networks $C = \{\text{CNN, MLP, HRNN}\}$. The networks, evident from their names, are based on three different network architectures– Convolution Neural Networks (CNN) [183], Multi-Layer Perceptrons (MLP) and Hierarchical Recurrent Neural Networks [184]– all of which give reasonably high accuracy on the two datasets (Table 7.1). For experiments on the ImageNET dataset, we use six pre-trained

	Legitimate User ($\mathcal{L}$)	Adversarial User ($\mathcal{A}$)							
MTDeep	Classification Image	FGM_m	FGM_c	FGM_h	DF_m	DF_c	DF_h	PGD_m	PGD_c	PGD_h
MLP	99.1	3.1	20.39	38.93	1.54	89.8	93.83	0.00	49.00	61.00
CNN	98.3	55.06	10.28	71.39	98.87	0.87	98.55	78.00	0.00	90.0
HRNN	98.7	25.12	27.24	11.43	95.38	83.17	3.66	23.00	51.00	0.00

(a) MNIST

	Legitimate User ($\mathcal{L}$)	Adversarial User ($\mathcal{A}$)							
MTDeep	Classification Image	FGM_m	FGM_c	FGM_h	DF_m	DF_c	DF_h	PGD_m	PGD_c	PGD_h
MLP	88.68	21.47	15.64	25.11	8.1	87.45	88.28	1.00	12.00	57.00
CNN	92.95	23.42	6.07	34.76	88.21	5.37	92.86	32.00	3.00	61.00
HRNN	89.16	29.44	43.53	14.85	74.9	87.64	9.57	41.00	60.00	0.00

(b) Fashion MNIST

Table 7.1: The actions of the players and the utilities of the two user types–$\mathcal{L}$ and $\mathcal{A}$ for (a) the MNIST and (b) the Fashion-MNIST datasets. The utility of the defender is exactly the same as that of $\mathcal{L}$ for the co-operative game against the player type $\mathcal{L}$ and a hundred minus the utility of $\mathcal{A}$ in the constant sum game against $\mathcal{A}$. The classification accuracy of each constituent network against the most effective attack, i.e. the worst case for each network, is highlighted in yellow.

networks that have won the image classifications over the last few years and have a reasonable high accuracy on the data-set (see Table 7.2). It is worth emphasizing that this ensemble of classifiers does not behave like well-known voting based ensembles and at classification time, uses only a single network's decision. [2] Formally, a pure strategy for the defender corresponds to selecting a single constituent network in the ensemble for each test input and use it for classification. A mixed strategy is a probability distribution over the different pure strategies and on every test input, the defender rolls a die (that represents the mixed strategy) to determine the constituent network it will use for classification.

[2] As we will later see, they can be viewed as a randomized ensemble or a stochastic classifier.

	$\mathcal{L}$	Adversarial User ($\mathcal{A}$)					
MTDeep	Image	UP_{VGG-F}	UP_{Caffe}	UP_{GoogLe}	UP_{VGG-16}	UP_{VGG-19}	UP_{Res}
VGG-F [185]	(92.9, 92.9)	(6.3, 93.7)	(28.2, 71.8)	(51.6, 48.4)	(57.9, 42.1)	(57.9, 42.1)	(52.6, 47.4)
CaffeNet [186]	(83.6, 83.6)	(26.0, 74.0)	(6.7, 93.3)	(52.3, 47.7)	(60.1, 39.9)	(60.1, 39.9)	(52.0, 48.0)
GoogLeNet [187]	(93.3, 93.3)	(53.8, 46.2)	(56.2, 43.8)	(21.1, 78.9)	(60.8, 39.2)	(60.2, 39.8)	(54.5, 45.5)
VGG16 [188]	(92.5, 92.5)	(36.6, 63.4)	(44.2, 55.8)	(43.5, 56.5)	(21.7, 78.3)	(26.9, 73.1)	(36.6, 63.4)
VGG19 [188]	(92.5, 92.5)	(36.0, 64.0)	(42.8, 57.2)	(46.4, 53.6)	(26.5, 73.5)	(22.2, 77.8)	(42.0, 58.0)
ResNet-152 [189]	(95.5, 95.5)	(53.7, 46.3)	(53.7, 46.3)	(49.5, 50.5)	(53.0, 47.0)	(54.5, 45.5)	(16.0, 84.0)

Table 7.2: Normal form game matrices for the defender and the User types $\mathcal{A}$ and $\mathcal{L}$ in the ImageNET scenario against Universal Perturbation attacks. The worst case classification accuracy of each constituent networks is highlighted in yellow. Similar to Table 7.1, we notice that the attacks developed against a particular network is the most effective attack against that network.

The Users ($\mathcal{L}$, $\mathcal{A}$)

The second player, in this game, is the user of the classification system. We divide the user into two-player types–Legitimate User ($\mathcal{L}$) and the Adversary ($\mathcal{A}$). $\mathcal{L}$ tries to input non-perturbed images (i.e. sampled from the expected test-set distribution that is represented in the training-set data) to the MTDeep system for classification. Given this is their only pure-strategy, $|A^{\mathcal{L}}| = 1$. The second type is the adversary $\mathcal{A}$ who essentially tries to perturb input images such that the classification system misclassifies these inputs. In our threat model, we consider a strong adversary who knows the different architectures we use in our MTDeep system. This means they can easily generate powerful white-box attacks for each of the networks in our system. We let $A^{\mathcal{A}}$ denote this set of attacks the attacker can generate against our system. For MNIST and Fashion-MNIST, we consider three classes of white-box attacks– the Fast Gradient Method (FGM), the DeepFool (DF) attack and the Projected Gra-

dient Descent. [3] (PGD) attacks (see Table 7.1) while for ImageNET, we restrict ourselves to universal perturbation attacks because the cost of constructing adversarial perturbations for each test image is computationally intensive (see Table 7.2). Each attack ($a \in A^{\mathcal{A}}$) is generated to cripple a particular constituent network in the MTDeep ensemble (indicated using the sub-script) but used against each of the defender's constituent networks. They may or may not be equally effective for all the configurations. In fact, for most the white-box attacks such as FGM and PGD, we show that although they have some *transferability* across the different networks, none can completely cripple all the networks (Table 7.1 and 7.2). MTDeep, as we will later see, leverages this fact to boost the security against adversarial examples.

Bayesian Game Formulation

MTDeep randomly picks a network n ($\in A^{\mathcal{D}}$) each time to classify an input image. If we use a naive switching strategy, such as uniform random selection, to pick a network whenever input is provided, we will have equal chances of choosing networks that have (1) low classification accuracy or (2) high vulnerability to perturbed images thereby being sub-optimal. Also, the attacker might eventually infer the defender's switching strategy and exploit the highly vulnerable configurations more often. Thus, reasoning about the optimal strategy is important. Thus, we model the interaction between the ensemble and the users as a Bayesian normal-form game, similar to our game-theoretic model in Chapter 3.

Existing works such as adversarial training design defense methods against adversarial attacks for DNNs formulate the problem as a zero-sum game where the attacker

[3]The the reward for PGD attacks are whole numbers because, in order to compute attack-based perturbations in a reasonable amount of time, we evaluate its effectiveness for perturbing 100 samples (as opposed to 10000 in the case of FGM and DF).

tries to maximize the defender's loss function by coming up with perturbed test examples that the network misclassifies, whereas the defender tries to reduce the loss on these adversarially perturbed examples. Given these methods are defined for a single neural network n, we can formally represent the objective as follows.

$$\min_{n} \max_{u} L_n(x+u, y)$$

where u represents the adversarial perturbation and L represents the loss function. Simply fine-tuning the classifier to have high accuracy on adversarially perturbed inputs often has the side effect of reducing the classification accuracy on non-perturbed inputs from the test set [161]. In this paper, we move away from the zero-sum game assumption and try to ensure that the defender minimizes the loss functions for both types of inputs images– original test set images and the adversarially perturbed ones. Formally, this would imply considering an optimization objective of the following kind for the network n,

$$\min_{n} \max_{u} L_n(x+u, y) + L_n(x, y)$$

Thus, we want MTDeep to be effective for $\mathcal{L}$ (denoted by the second term and proportional to minimizing the loss on the original test set) and, at the same time, increase the accuracy of classification for the perturbed images (denoted by the first term and proportional to minimizing the loss against adversarial inputs at test-time), making this a multi-objective optimization problem.

A natural question that arises is how much importance should we associate with the two different objectives. If we only train on adversarially perturbed images, we lose accuracy on test inputs and vice-versa. Hence, a trade-off is necessary. We can choose to represent this trade-off using a parameter α and change the objective

function defined above as follows.

$$\min_{n} \max_{u} \alpha \cdot L_n(x+u, y) + (1-\alpha) \cdot L_n(x, y) \tag{7.1}$$

First, we want to highlight that α may often be highly application-specific. For example, a banking system that uses handwritten digit recognition for identifying monetary amount on bank cheques should prioritize maximizing accuracy on adversarial examples over an occasional misclassification on the actual test set, whereas, an image captioning system that is trying to help a visually challenged person understand posts on social media hardly needs to care about adversarial examples. Second, in the context of an ensemble, as opposed to a single-network n, this trade-off α represents the probability of the defender's belief about whether a particular input at test time is drawn from the adversarially perturbed set (and $(1-\alpha)$ represents the probability of it being drawn from the legitimate test set), making this a Bayesian game. We can now define the utility of the player in this game to act as a stand-in for the $L_n(x+u, y)$ and $L(x, y)$ in Equation 7.1.

- The Legitimate User ($\mathcal{L}$) and the defender both get a reward value that represents the accuracy of the DNN system (a stand in for $L(x, y)$. Thus, for using a network n in the ensemble N with classification accuracy (say) 98% for an input image both the defender and $\mathcal{L}$ get a reward of 98 (see Table 7.1 & 7.2).

- The Adversary ($\mathcal{A}$) and the defender play a constant(= 100) sum game, where the former's reward value for an attack a against the network n is given by $e_{n,a}$, which is the *fooling rate* and the defender's reward is the accuracy of n on perturbed inputs, which is $(100-e_{n,a})$ (see Table 7.1; a stand-in for $L_n(x+u, y)$).

Similar to the MTDs discussed, here the defender $\mathcal{D}$ deploys a system upfront; thus, the adversary $\mathcal{A}$ can perform reconnaissance before attacking it. This imparts the leader-follower paradigm to our formulated Bayesian Game. Hence we seek to find a mixed strategy for the defender. In terms of the optimization problem defined for the single-network setting in the last section, the objective becomes the following for the ensemble (considering a single attack action u for the attacker).

$$\min_{x} \max_{u} \alpha \cdot x_n L_n(x+u, y) + (1-\alpha) x_n \cdot L_n(x, y) \tag{7.2}$$

where x denotes the mixed strategy for the defender and thus, x_n represents the probability with which they choose the network $n \in N$ for classification. Given that we consider the accuracy values, that are inversely proportional to the loss values in Equation 7.2, the defender seeks to maximize this objective. With the consideration of multiple attacks for the adversary and the inherent leader-follower paradigm, this optimization boils down to finding the Stackelberg Equilibrium of the game. Let us denote the strategy vector for the defender as $\vec{x}$ and their reward as $R^{\mathcal{D}}_{n,a}$ when the defender uses the network n and user selects the action a. Similarly, the strategy vectors for the adversary and the legitimate user types are $\vec{q}^{\mathcal{A}}$ and $\vec{q}^{\mathcal{L}}$ and their rewards are $R^{\mathcal{A}}_{n,a}$ and $R^{\mathcal{L}}_{n,a}$ respectively. We seek to maximize the defender's reward while allowing the attacker to choose the most effective attack. Specifically, we solve the following optimization problem –

$$\begin{aligned} \max_{x,q} \sum_{n \in N} (\alpha \cdot \sum_{u \in U} R^{\mathcal{D}}_{n,a}\ x_n q^{\mathcal{A}}_u \quad &+ \quad (1-\alpha) \cdot R^{\mathcal{D}}_{n,a}\ x_n q^{\mathcal{L}}_u) \\ \text{s.t.} \qquad \sum_{n \in N} x_n \quad &= \quad 1 \end{aligned}$$

$$
\begin{aligned}
\sum_{u \in U} q_u^{\mathcal{Y}} &= 1 \qquad \forall\, \mathcal{Y} \in \{\mathcal{A}, \mathcal{L}\} \\
0 \leq\ x_n &\leq 1 \qquad \forall\, n \in A^{\mathcal{D}} \\
q_u^{\mathcal{Y}} &\in \{0,1\} \qquad \forall\, \mathcal{Y} \in \{\mathcal{A}, \mathcal{L}\} \\
0 \leq v^{\mathcal{Y}} - \sum_{n \in N} R_{n,a}^{\mathcal{Y}} x_n &\leq (1 - q_a^{\mathcal{Y}})M \quad \forall\, u \in U^{\mathcal{Y}},\ \forall\, \mathcal{Y} \in \{\mathcal{A}, \mathcal{L}\}
\end{aligned}
$$

where α is the probability of $\mathcal{A}$ attacking an MTDeep system and M is a large positive number. The objective function maximizes the defender's expected reward over its own switching strategy $\vec{x}$ and the strategy vector played by the two user types $(\vec{q}^{\mathcal{A}}, \vec{q}^{\mathcal{L}})$ weighted by their relative importance α, which is the probability with which the defender expects the attacker type $\mathcal{A}$ attacks their system.

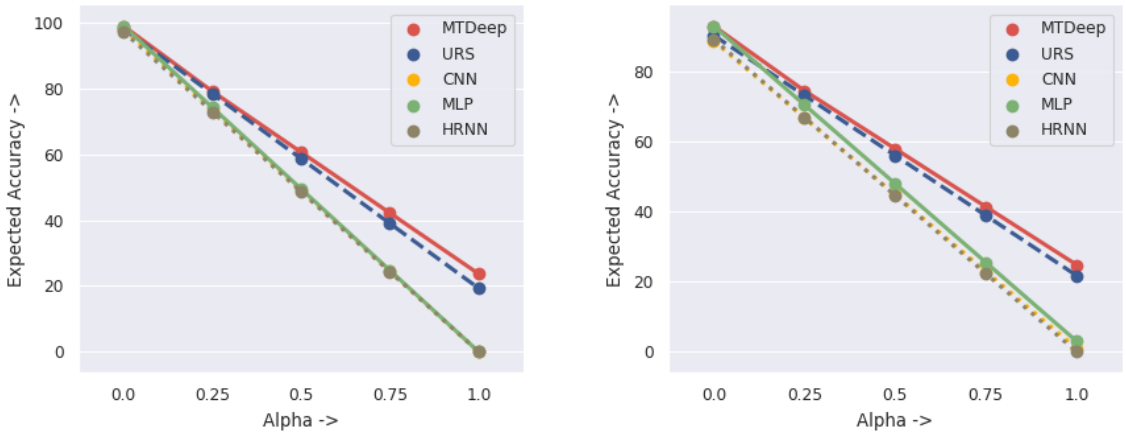
Thus, this MIQP, implemented in Gurobi, takes as input (1) the reward values $R^D, R^{\mathcal{A}}$, and $R^{\mathcal{L}}$ obtained from accuracy metrics of the constituent networks, (2) α, the probability of the player types and outputs the optimal strategy for both the defender $(\vec{x})$ and the users. The first four constraints ensure that the strategy vectors sum up to one since they represent the probability of selecting actions. The fifth constraint represents the dual of the attacker's optimization problem which tries to maximize their expected reward $v^{\mathcal{Y}}$ over the defender's strategy. This constraint captures the fact that the attacker knows $\vec{x}$ and uses it to select its attack strategy $\vec{q}^{\mathcal{A}}$. Note that the second constraint forces the users $\mathcal{L}$ and $\mathcal{A}$ to select a pure strategy. As the authors in [103] show, this constraint is not limiting for the attacker because for the attacker $\mathcal{A}$, there always exists a pure strategy in support of any mixed strategy it can play. For the attack the attacker selects, the right side of the fifth constraint becomes 0 making $v^{\mathcal{Y}} = \sum_{n \in N} R_{n,a}^{\mathcal{Y}}$. Lastly, the defender's strategy, in the worst case, can be a pure strategy that directs MTDeep to use a single network for classification.

7.4 Experimental Results

We first compare the effectiveness of MTDeep for MNIST, Fashion-MNIST, and the ImageNet datasets when it is used as a standalone defense mechanism. We then show that MTDeep when piggybacked onto an existing defense mechanism like Ensemble Adversarial Training, can result in boosting the accuracy against adversarial attacks. We then analyze the effect of black-box attacks designed using a distilled network that can capture a holistic view of an ensemble like MTDeep which leverages randomization at test time. We show that given the limitation on the number of samples in the MNIST dataset, black box attacks are less effective than white-box attacks. We finally introduce the notion of differential immunity and show that this metric can capture the informal notion of transferability of attacks. We discuss how this measure can give us an understanding of how effective MTDeep will be. Finally, we talk about the effects of setting an incorrect α (that a user needs to input) when calculating the switching strategy for MTDeep.

7.4.1 MTDeep as a Standalone Defense Technique

We compare the effectiveness of MTDeep with two baselines. The first one measures the accuracy of each individual network in the ensemble and the second one is a randomized ensemble that uses Uniform Random Strategy (MTD-URS) to pick one of the constituent networks with equal probability. In contrast, MTDeep uses the Stackelberg equilibrium strategy of the defender to pick a constituent DNN at random. We do not showcase comparison against deterministic (such as majority-voting or weighted) ensembles because, as discussed in the related work section (Sec. 7.1), these voting functions are equivalent to a final layer of a large network with multiple sub-components built using CNN, RNN and MLP building blocks. Nonetheless,



(a) MNIST

(b) Fashion-MNIST

Figure 7.2: Accuracy of MTDeep with non-adversarially trained networks against (1) each of the constituent networks and (2) a uniform random strategy for randomly selecting a constituent network at test time. The gray line at the 10% mark denotes the accuracy of randmonly guessing a class given an input image.

to drive the point home, under the heading of differential immunity, we empirically demonstrate that majority voting ensembles obtain a lower accuracy on adversarial examples when compared to MTDeep (and even MTD-URS) for MNIST.

MNIST and Fashion-MNIST

For each of the data sets, we trained three classification networks that, as stated before, were built using either Convolution layers (CNN), Multi-layer Perceptrons (MLP), or Hierarchical Recurrent layers (HRNN). The size of the train and test sets were 50000 and 10000 respectively.

We considered three attack methods for the attacker– the Fast Gradient-Based (FGM) attack (with $\epsilon = 0.3$), the DeepFool (DF) attack (with three classes being

considered at each step when searching for an attack perturbation), and the Projected Gradient Descent (PGD) attack (with $\epsilon : 0.3$, $\epsilon - iter : 0.05$). We develop adversarial examples for each test image based on either the loss gradient (for FGM and PGD) or the classification boundary in the feature space (for DF) corresponding to each individual network in the ensemble. For example, the adversarial examples generated using the PGD algorithm on the loss information of the CNN is termed as PGD_c in Table 7.1. We then find the classification accuracy of each network on these adversarial examples to compute the utility values shown in the table. Note that an adversarial example developed using information about one network may not be as effective for the other networks. We find that this is especially true for attacks like DF that exploit information about a particular network's classification boundary (eg. DF_m reduces the classification accuracy of MLP to 2% but is hardly effective against the other two networks. Both of these are able to classify the adversarial examples correctly more than 95% of the time). On the other hand, attacks that exploit the gradient signals of a particular network are somewhat effective against the other networks, i.e. have high transferability (eg. FGM_m reduces the accuracy of MLP to 3.1% and the accuracy of HRNN and CNN to $\approx 25\%$ and $\approx 55\%$ respectively). We observe this trend for both the MNIST and the Fashion-MNIST dataset.

In Figure 7.2, we plot the accuracy of a particular classification system (the objective function value), when using MTDeep *vs.* any of the single constituent networks and MTD-URS as α varies from 0 to 1. When $\alpha = 0$ and the defender ignores the possibility of playing against an adversary, and thus, the mixed strategy for MTDeep boils down to a pure strategy for selecting the most accurate classifier. In our experiments, MTDeep chooses the MLP for every input test-image for the MNIST data-set and the CNN for classifying inputs drawn from Fashion-MNIST. In contrast, MTD-URS has lower classification accuracy than MTDeep because it also uses the two less

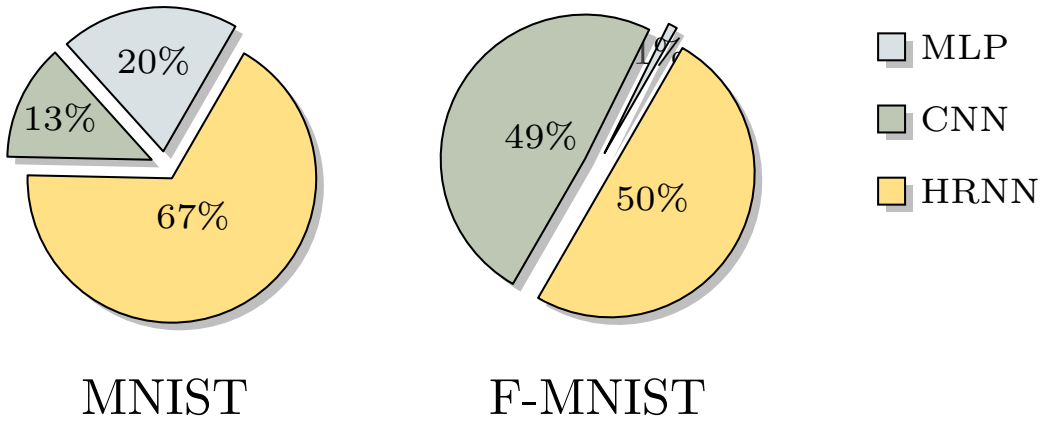


Figure 7.3: MTDeep's classifier selection strategy.

accurate classifiers equal amounts of time. Given that classification accuracy for each of the constituent networks are relatively high, the difference is hard to notice in the graph.

When $\alpha = 1$ and the defender cares about accuracy on only adversarial examples, strong attacks like PGD for a particular network can fool it 100% of the time for MNIST data classification and at least 97% for Fashion-MNIST. In contrast to using individual networks, a randomized selection of networks at classification time perform much better because an adversarial perturbation developed based on information from one network fails to fool other networks that may be selected at classification time. MTDeep achieves a classification accuracy of 24% for MNIST and 25% for Fashion-MNIST while MTD-URS has a classification accuracy of $\approx$ 20% for both the data sets. The difference in classification accuracy is mainly because MTD-URS picks more vulnerable networks with equal probability. The mixed strategies for MTDeep are show in Figure 7.3.

Note that in the case of Fashion-MNIST, MLP has a very low probability of being played ($\approx 0.001\%$) and the classification system is found to be the most secure when

utilizing a subset of two consequent networks (i.e. CNN and HRNN). On the other hand, for the classification of MNIST data, the MLP has a higher probability of being played at equilibrium than the CNN-based classifier. HRNN is given equal weight as CNN for Fashion-MNIST but clearly dominates in the case of MNIST.

ImageNET

We use six different networks which have excelled on ILSVRC-2012's validation set [190] to construct the ensemble for MTDeep (see Table 7.2). Since attacks like FGM, DF, and PGD on these large networks have are time intensive because they need to be calculated for every single test image, we assume the adversary uses Universal Perturbations (UP) developed for each network in [172], which (1) is built on top of DF and (2) have to be generated only once. These UPs were generated by ensuring that the L_∞ norm of the perturbations was less than a bound $\xi = 10$. The actions of the players and their utilities are shown in Table 7.2.

Researchers have shown that defense mechanisms like adversarial training are ineffective against this type of attack [172]. Moreover, state-of-the-art defense mechanisms (c.f. discussion in related work), is still ineffective against this attack. In such cases, MTDeep is a particularly attractive approach because it can increase the robustness of the classification system even when all other defense mechanisms are ineffective.

In Figure 7.4, we plot the expected accuracy for the MTDeep along with the objective values of each of the constituent networks when the probability of an adversary type α varies. Given there are six constituent networks in the ensemble, to avoid clutter, we don't plot MTD-URS for brevity but observe that it always has $\approx 4\%$ less accuracy than MTDeep, which is a relatively high drop in accuracy given the Ima-

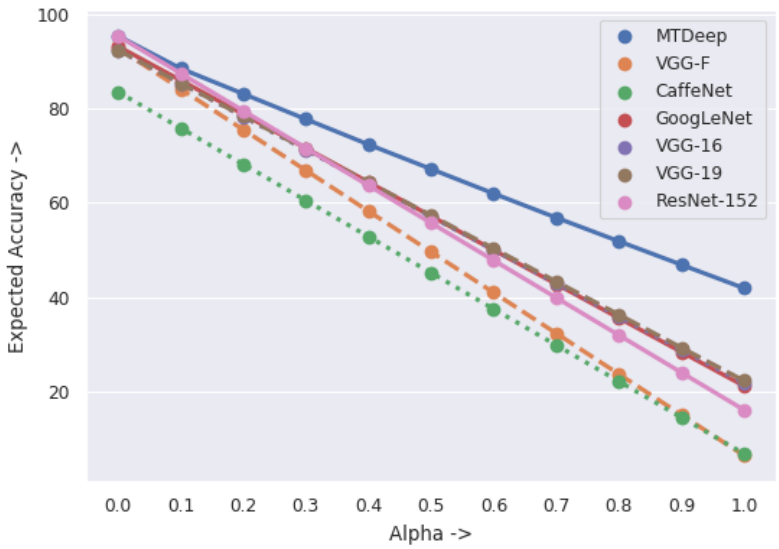


Figure 7.4: We compare the expected accuracy of MTDeep against the accuracy of the individual constituent networks for the ImageNET dataset.

geNET data-set. When $\alpha = 0$, MTDeep uses the most accurate network (ResNet-152) that maximizes the classification accuracy. As adversarial inputs become more ubiquitous and thus α moves towards 1, the accuracy against the perturbed inputs drops for all the constituent networks of the ensemble. Thus, to stay protected, MTDeep switches to a mixed policy that utilizes more networks.

When the system receives only adversarial samples, i.e. $\alpha = 1$, the accuracy of MTDeep is 42% compared to 20% for the best of the single DNN architectures. The optimal strategy in this case is $\vec{x} = (0, 0.171, 0.241, 0, 0.401, 0.187)$ which discards some of the configurations ($VGG\text{-}F$ and $VGG\text{-}16$ in this case). The 22% accuracy bump for modified images comes despite (i) high misclassification rates of constituent networks against Universal Perturbations, and (ii) lack of proven defense mechanisms against such attacks.

	Legitimate User ($\mathcal{L}$)	Adversarial User ($\mathcal{A}$)								
MTDeep	Classification Image	FGM_m	FGM_c	FGM_h	DF_m	DF_c	DF_h	PGD_m	PGD_c	PGD_h
MLP_{eat}	97.99	95.06	75.32	70.1	1.5	96.97	95.73	0.00	88.00	69.00
CNN_{eat}	98.97	61.44	96.55	68.58	98.36	0.79	96.09	72.00	20.00	81.00
HRNN_{eat}	97.22	81.24	84.79	93.1	96.85	95.9	4.41	82.00	71.00	10.00

Table 7.3: The utilities for the players when the adversary attacks the classifiers already hardened using Ensemble Adversarial Training (EAT) with FGM attacks.

Remark. Let us denote accuracy on legitimate samples as $a_{\mathcal{L}}$ and accuracy on adversarial samples as $a_{\mathcal{A}}$. Note that the objective function (O), becomes the equation of a line when $a_{\mathcal{L}}$ and $a_{\mathcal{A}}$ are constants because $O = (a_{\mathcal{A}} - a_{\mathcal{L}}) * \alpha + a_{\mathcal{L}}$. Since the values of $a_{\mathcal{L}}$ and $a_{\mathcal{A}}$ are constant for each constituent network, the expected accuracy ($= O$) results in a straight line with slope $(a_{\mathcal{A}} - a_{\mathcal{L}})$ and intercept $a_{\mathcal{L}}$. Also, as the accuracy on the legitimate samples is more than accuracy on the adversarial inputs, i.e. $a_{\mathcal{L}} > a_{\mathcal{A}}$, the slope is negative. For the MTDeep system (and also MTD-URS), the change in the accuracy values $a_{\mathcal{A}}$ and $a_{\mathcal{L}}$ is small ($2 - 4\%$ relative to the 100% scale of Y-axis) as α varies from 0 to 1. Thus, the plots although non-linear, at times *appear* to be linear.

7.4.2 MTDeep as an Add-on Defense-in-depth solution

We study the use of MTDeep on top of a state-of-the-art defense mechanism called Ensemble Adversarial Training (EAT) [166]. EAT is an improvement on top of the adversarial training procedure in which (1) an attack algorithm is chosen, (2) perturbed images are generated using it for a particular network and (3) the generated data is used (with their correct labels) to fine-tune the weights of the trained network that needs to be made more robust. Although this helps to robustify the network to an extent, higher gains in accuracy against adversarial examples can be gained by incorporating more perturbed examples in the new test set, especially the ones

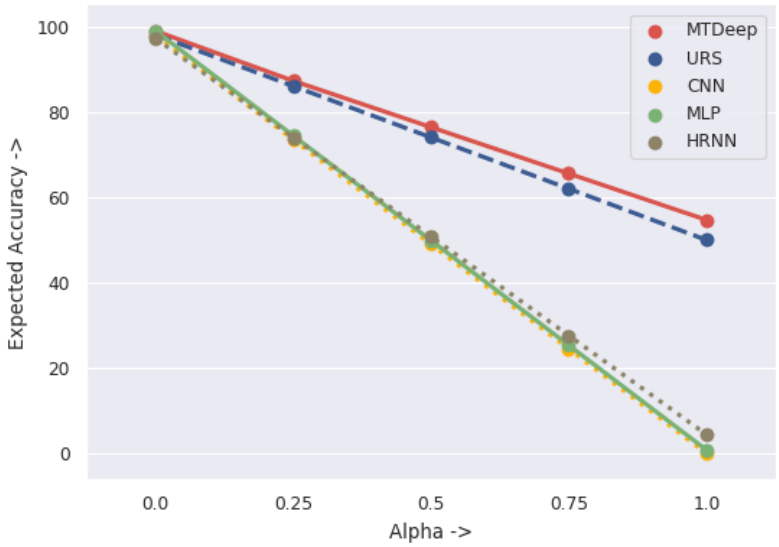


Figure 7.5: Expected accuracy of MTDeep with adversarially trained networks.

that are generated by attacking other networks (i.e. use all, not only the one whose parameters will be fine-tuned). As more than one network is required in this defense procedure, the authors call this as Ensemble Adversarial Training even though the end product of this procedure is a single network more robust to adversarial attacks. Note that MTDeep renders itself naturally to this robustification method and also, with high probability, uses all the robust constituent networks in the ensemble at test time.

Unfortunately, using EAT can only make the networks robust against attack images generated by the particular attack algorithm used for fine-tuning and may still be vulnerable to stronger (i.e. more computationally intensive) attacks. In Table 7.3, we show that the utility values obtained using the three constituent networks whose parameters are fine-tuned using EAT (which, in turn, uses the FGM attack to generate training samples on top of the MNIST test set). Note that although there is a boost in overall accuracy against adversarial examples generated using FGM, the other attacks (1) DF, which is generated in a very different manner compared to

FGM, and (2) PDG, which represents a stronger class of attacks, are both still able to cripple the individual constituent networks. Surprisingly, even for these attacks, the EAT procedure increases the accuracy of attacks that are *misaligned*. For example, an attack PGD_H generated using the model parameters of the HRNN_{eat} brings down the accuracy of the HRNN_{eat} network to 9% whereas, it is found to be pretty ineffective against the CNN_{eat} ($\approx 81\%$) and the MLP_{eat} ($\approx 72\%$). As to why EAT helps is reducing the transferability of these attacks could be interesting future work. In the present context, this phenomenon helps MTDeep used in conjunction with the EAT method to obtain impressive accuracy gains against attack images.

We highlight the results of our experiments with the fine-tuned networks on the MNIST dataset in Figure 7.5. When $\alpha = 1$, i.e. the worst case for the defender and it only gets adversarially perturbed images as inputs, the accuracy of the constituent networks is $0 - 4\%$ because the EAT training is using the FGM attack is ineffective against DF and PGD attacks for a particular network. On the other hand, MTDeep achieves an accuracy of $\approx 55\%$ against adversarially perturbed images because of the reduced effectiveness in terms of the transferability of the attack images. Thus, we see a gain of more than 50% when classifying only adversarially perturbed images.

7.4.3 Blackbox Attacks on MTDeep

MTDeep designs a strategy based on a set of known attacks. Once deployed, an attacker can train a substitute network via distillation, i.e. use MTDeep as an oracle to obtain labels for the (chosen-ciphertext like) training set for the substitute network. Given that the distilled network captures information relating to the randomization at test time, we wanted to see how effective such a distillation procedure is in generating an expected network that mimics MTDeep. More specifically, if adversarial samples

generated on this distilled network [164] successfully transfer against the MTDeep ensemble.

For this purpose, we used the non-adversarially trained networks for classifying MNIST data and consider the worst-case scenario where all inputs at the test time are adversarially modified, i.e. $\alpha = 1$. Note that a distilled network needs to capture both (1) the behavior of the ensemble and (2) the built-in randomization (expected classification boundary) of the MTDeep ensemble with limited training samples (50000, which is equal to the size of the training set for the constituent networks) in order to be effective. We notice that MTDeep has higher immunity to black-box attacks and is able to classify attack inputs $\approx 32\%$ of the time compared to the $\approx 24\%$ accuracy against white-box attacks, as discussed in the previous sub-section. Thus, there exists a white-box attack in the attacker's arsenal that is stronger than the black-box attack we generated, thereby not affecting the defender's optimal mixed strategy.

Note that even if a black-box attack proved to be a more effective attack against the ensemble (which it may be for some other domain or vision dataset), this attack is not modeled by the defender in the original game. The defender with knowledge of such black-box attacks can do two actions– (1) incorporate the black-box attack as one of the attacker's actions which in turn, might change the mixed strategy for random selection of constituent networks and (2) train the individual networks against adversarial images generated by this attack. Both of these can, in turn, lead the attacker to come up with new black-box attacks against the improved ensemble. As to how and when, if at all, this procedure leads to a stable point is another interesting future research direction.

Networks	Differential Immunity (δ)	Accuracy of Best Constituent Net	Accuracy of MTDeep	Gain
FashionMNIST	0.11	3%	24.8%	21.8%
MNIST	0.19	0%	23.68%	23.68%
ImageNET	0.34	22.2%	42.88%	20.68%
MNIST + EAT	0.78	4.41%	54.71%	50.3%

Table 7.4: Differential Immunity of the various ensembles and the gains seen in accuracy compared to the best constituent networks when $\alpha = 1$.

7.4.4 Differential Immunity

Clearly, when an attack $u \in U$ is able to cripple all the networks $n \in N$, using MTDeep will provide no gains in robustness. In this section, we try to quantify the gains MTDeep can provide. Let $E : N \times A^{\mathcal{A}} \rightarrow [0, 100]$ denote this fooling rate function where $E(n, a)$ is the fooling rate when an attack a is used against a network n. Differential immunity of an ensemble U against a set of known attacks E against it δ can measured with just the fooling rate values as follows,

$$\delta(U, N) \quad = \quad \min_{u} \frac{\max_n E(n, a) - \min_n E(n, a) + 1}{\max_n E(n, a) + 1}$$

If the maximum and minimum fooling rates of a on a N differ by a wide margin, then the differential immunity of MTDeep is higher. This is represented in the numerator. The denominator ensures that an attack that has a high fooling rate reduces the differential immunity of a system compared to a low impact attack even when the numerator is the same. The $+1$ factor in the denominator of the function prevents division by zero while the $+1$ in the numerator ensures that higher values of $\max_n E(n, a)$ reduce the δ when $\max_n E(n, a) = \min_n E(n, a)$. Note that $\delta \in [0, 1]$.

As per this measure, the differential immunity of the various ensembles used in our experiments is highlighted in Table 7.4.

As per our expectation, we observe a general trend that the differential immunity of an ensemble is in proportional to the accuracy gains obtained by MTdeep when compared to the most secure constituent network in the ensemble. Although we notice the lowest gain in case of ImageNET, note that this 20.68% is a substantially better absolute gain in accuracy than the $\approx 22\%$ or the $\approx 24\%$ gain in accuracy for the Fashion-MNIST and the MNIST datasets with non-adversarially trained DNNs because the number of classes in ImageNET is 1000 compared to 10 for the latter two datasets. A random class selector with zero understanding of the input (provided there is no class imbalance among the adversarial inputs) can achieve $\approx 10\%$ accuracy for MNIST and Fashion-MNIST whereas it can only obtain an accuracy of $\approx 0.001\%$ for the ImageNET data-set.

Note that existing measures of robustness are mostly designed for a single DNN [181, 182] and thus, do not try to incorporate the notion of transferability, i.e. to what extent is an attack designed for one network can affect another. Thus, they cannot be used to correctly measure the robustness of an ensemble. We propose *differential immunity* as one of the metrics for evaluating ensembles that use any form of randomization at test time. It can be used to capture the transferability of an adversarial attack and thus, provide a reasonable measure of robustness for ensembles.

Disagreement Metrics In Fig. 7.5, we highlight the number of perturbed test images (total 10000) on which $0, 1, 2$ or 3 constituent DNN's classification output(s) agree with the correct class label. We conducted these experiments using the non-adversarially trained networks for MNIST classification and for brevity purposes, we

Attacks	0	1	2	3
FGM_C	4788	3641	1449	118
FGM_H	389	2728	6667	212
FGM_M	1513	5790	2479	214
FGM_{BB}	2305	2569	2678	2444

Table 7.5: Agreement among constituent networks when classifying perturbed inputs for the MNIST data-set.

only use the FGM attack method. Note that the FGM_C is the strongest attack that can make all the $n \in N$ misclassify at least 70% of the images. As generating δ can be costly at times, which needs the fooling rates for each pair (n, a), one can generate the agreement metrics on a small data set to provide upper bounds for δ. This provides an idea as to how using an MTDeep ensemble can increase the robustness against adversarial samples. In this case, $\delta_{MNIST} \leq 0.51$ because for the strongest attack, every network in the ensemble will misclassify (approx.) 49% of the time. Also, note that a majority based ensemble is only able to guarantee an accuracy of $\approx 14\%$ against the FSM_C attack because, in all the other cases, only 0 or 1 network is able to correctly predict the correct class. In comparison, MTDeep when facing an attacker who only uses FGM attacks obtains an accuracy of 26.8% against adversarially perturbed inputs.

Towards Differentially Immune Networks. Previously, authors in [6] have shown that constructing an ensemble with a high δ is difficult. The authors show that ideas like partitioning the training data into disjoint sets that are then used to train different networks ($\in A^{\mathcal{D}}$) do not make the networks differentially immune. This

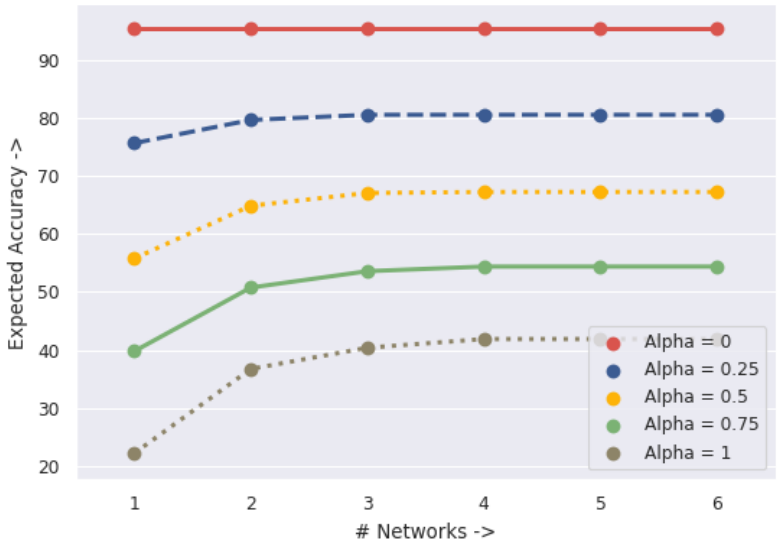
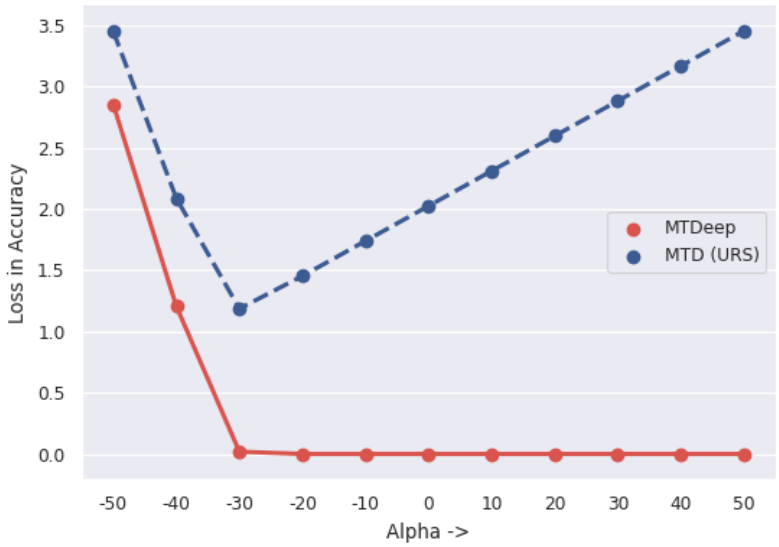


Figure 7.6: Analyzing the participation of the different constituent networks (trained on ImageNET) at equilibrium in MTDeep for different values of α.

concept of an attack's potency across networks it was not specifically targeted for is defined as transferability of an attack [6] and, although informally used, is similar to our notion of differential immunity. Fortunately, recent works highlight promising avenues that can be used to limit the transferability of attacks [180]. If an ensemble of such networks can be developed, as we saw in the case of DNNs for MNIST fine-tuned with EAT, MTDeep can provide significant gains as a defense technique (further discussion on this topic ensues in Appendix C). In scenarios where generating ensembles with high differential immunity is still difficult, MTDeep can still boost the accuracy of classifiers (in conjunction or without other state-of-the-art defense mechanisms).

7.4.5 Participation of Individual Networks.

In Figure 7.6, we explore the participation of individual networks in the mixed strategy equilibrium for MTDeep used to classify ImageNET data. The results clearly



show that while it is useful to have multiple networks providing differential immunity (as testified by the improvement of accuracy in adversarial conditions), the leveling-off of the objective function values with more DNNs in the mix does underline that there is much room for research in actively developing DNNs that can provide greater *differential immunity*. Note that no more than four (out of the six) networks participate in the equilibrium. An ensemble of networks with higher differential immunity equipped with MTD can thus provide significant gains in both security and accuracy.

7.4.6 Robustness against Miscalibrated α

So far in our discussion, we have assumed that α (the attacker's probability) is calibrated correctly when coming up with a randomization strategy. Similar to our discussion on the robustness of the strategy found to attacker probabilities in Chapter 3, the value of α may be incorrect and the computed strategy, in turn,

ends up being sub-optimal. In Figure 7.7, we plot the deviation of the chosen policy (based on the assumed α) from the optimal as the real α is varied $\pm 50\%$ from the one assumed. The BSG-framework remains quite robust (as opposed to a uniform random strategy) i.e. the accuracy is within $0 - 3\%$ of the optimal accuracy. The robustness to α further highlights the usefulness of MTDeep as a meta-defense meant to work not only against adversarial attacks but also in the context of a deployed classifier that will have to deal with adversaries as well as legitimate users.

7.5 Concluding Remarks

In this chapter, we introduced MTDeep – a framework inspired by Moving Target Defense in cybersecurity (a 'security-as-a-service')– to help boost the security of existing classification systems based on Deep Neural Networks (DNNs). We modeled the interaction between MTDeep and the users as a Bayesian Stackelberg Game, whose equilibrium gives the optimal solution to the multi-objective problem of reducing the misclassification rates on adversarially modified images while maintaining high classification accuracy on the non-perturbed images. We empirically showed the effectiveness of an MTDeep ensemble of classifiers against various attacks on the MNIST, the Fashion-MNIST, and the ImageNet data-sets. When MTDeep is used along-side existing defense mechanisms for DNNs, it increases the gain in robustness. Finally, we discussed the property of differential immunity and highlighted its relation to the accuracy gains that can be obtained by using MTDeep; it makes the need for developing ensembles with higher differential immunity a promising future direction.

Chapter 8

MOVING TARGET DEFENSE FOR ROBUST SENSOR ACTIVATION IN POWER NETWORKS

C	Detection Surface Shifting of Differentially Immune Sensor Placement Strategies in Power Networks
t	Constant Period switching
M	Stackelberg Strategy of a Normal-form Game

The electric power grid forms the backbone of all the other critical infrastructures (communication, transportation, water distribution, etc) of a country, and thus, necessitates the presence of adequate monitoring strategies to quickly detect any anomalous behavior(s) that may have manifested in the system. It is of utmost importance to not only detect such anomalous behavior but also to take appropriate actions quickly to prevent the failures of power grid components which in turn, may lead to a large scale blackout [191]. Components such as High Voltage Transformers (HVTs), generating stations, substations, etc. are essential to the power grid and thus, their operational behaviors are monitored at all times with the help of Phasor Measurement Units (PMUs are devices, which are utilized as sensors, for monitoring the power grid). The problem of placing these sensors has been studied by multiple researchers over the past decade [192, 193]. Recently, in [194, 195], the authors proposed a sensor

placement approach that can *uniquely identify the source of the anomaly by utilizing the sensor readings generated by PMUs*. With the discovery of novel real-world attacks such as Stuxnet [196], Dragonfly [197] and an array of existing cyberattack such as jamming, Denial of Service, packet dropping, false-data injection and compromise of data integrity [198, 199], robustness of existing sensor placement mechanisms becomes critical. Thus, in this chapter, we seek to leverage the idea of Moving Target Defense (MTD) and Minimum Discriminating Code Sets (MDCS) in designing a defense-in-depth solution to PMU placement in adversarial environments.

We continuously move the detection surface to make it challenging for an adversary to impede the unique identification of failure signals of HVTs. While PMUs are difficult to move, as opposed to the movement of physical resources in security games [103], once placed, they can be efficiently activated and deactivated, similar to the dynamic movement of Intrusion Detection Systems (IDS) discussed in Chapter 4. While one may choose to activate all the PMUs placed upfront, the cost of maintaining them can become an impediment. Hence, the periodic use of a smaller subset (which ensures unique identification) of the sensors placed upfront can be considered. Further, all works in MTD have relied solely on heuristic guidance when constructing the configuration set; this can, as discussed in Chapter 7, result in scenarios where one attack cripples all defenses. The *goal of this chapter* is thus to construct a configuration set C, which has the desirable property of *differential immunity* introduced in Chapter 7. We then model the attack-defense interaction as a game and infer the optimal movement strategy M.

Specifically, we first define a novel variant of the MDCS problem, called the $K-$differentially Immune MDCS (hereafter called K-δMDCS). A solution to this problem is a set of K MDCSs of a graph, i.e. each of the K solutions uniquely iden-

tify failing HVTs, with the added constraint that no two MDCSs share a common vertex. We will show that K MDCS form a differentially immune configuration set for the MTD. Given that the original MDCS problem is NP-Complete, we show that K-δMDCS is also NP-Complete and provide an optimal Quadratically Constrained Integer Linear Programming (QC-ILP) approach to find the $K_{\max}$-MDCS of a graph. While our approach proves to be scalable for even the large power networks, we also propose a greedy approach that is computationally faster but trades-off on finding the maximum value of K. Second, we model the interaction between the power utility company (hereafter, the defender) and the adversary, as a normal-form game. Similar to our work described in the previous chapter, the Strong Stackelberg Equilibirum characterizes an optimal sensor activation strategy for the defender. Finally, we show the efficacy of our strategy and the scalability of our proposed approach on several IEEE power test cases with varying sizes.

8.1 Preliminaries

In this section, we first describe an electric power grid scenario and highlight how it can be modeled as a graph. Then, we describe the MDCS problem, showcasing how solutions to it can help determine sensor placement for unique monitoring of HVTs. Finally, we re-state the notion of differential immunity presented in the previous chapter.

The Electric Power Grid as a Graph

In Figure 8.1, we show the IEEE 14 Bus single line diagram of an electrical power grid. In [194], the authors proposed a set of graph construction rules that model the monitoring of HVTs as a bipartite graph $G = (T \cup S, E)$, where T represents the set of High Voltage Transformers (HVTs) that need to be uniquely monitored and

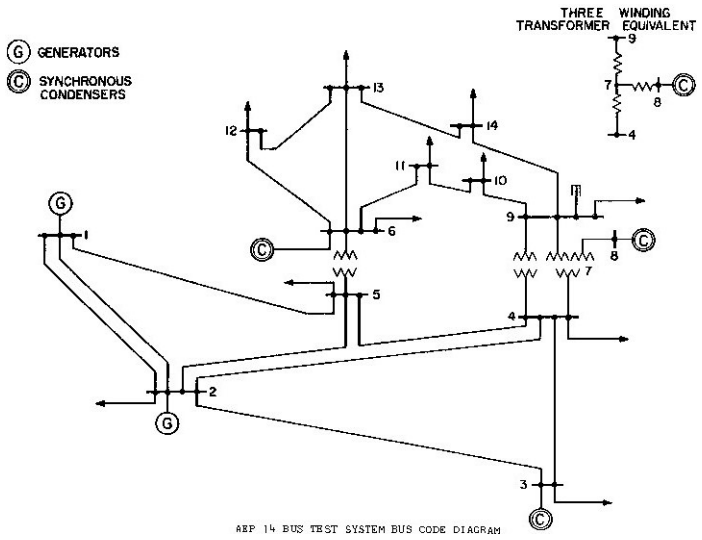


Figure 8.1: IEEE 14 Bus Single Line Diagram

S represents the locations where the PMUs (or sensors) can be potentially placed (PMU's cannot be directly placed on HVTs), and E represents the set of edges that exist if the operational behavior signal of an HVT ($t \in T$) reaches a PMU ($s \in S$) within a pre-specified number of hops. As Signal-to-Noise ratio (SNR) is used to measure the operational signal of an HVT in the real-world, and SNRs are known to quickly deteriorate over multiple hops, we, similar to prior works [194, 195], consider an edge ($\in E$) to be at most 2 hops (see Figure 8.2).

Minimum Discriminating Code Set (MDCS)

The MDCS problem is a special case of the Minimum Identifying Code Set (MICS) [200], and was first studied in [201]. Given a graph, the goal of MICS is to identify the smallest set of nodes on which sensors can be placed such that two properties are met (given domain-specific information propagation constraints). First, if an event occurs in an entity represented by a node in the graph, a unique set of sensors is activated leading to easy identification of the node (entity). Second, all

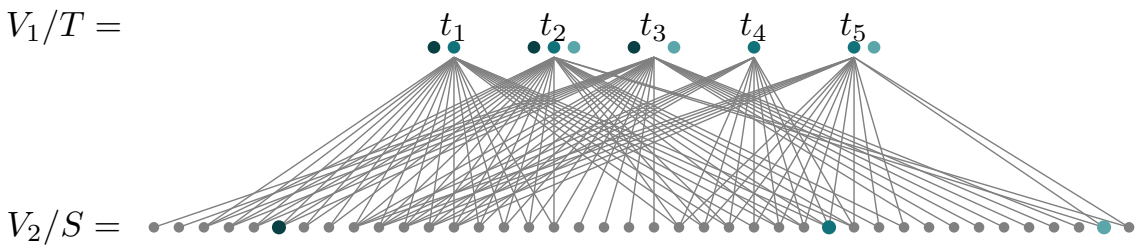


Figure 8.2: Bipartite Graph derived from the IEEE 14-bus network with 2-hop information propagation constraints.

nodes should trigger a non-empty set of sensors if an event occurs. In MDCS, the problem is adapted to a bipartite graph scenario with two (disjoint) sets of nodes– (i) nodes of interest where an event may occur; these nodes have to be uniquely identified with the sensors, and (ii) nodes on which sensors can be placed. Formally, we can define the MDCS problem in the context of sensor placement in power grid systems as follows [194].

Definition 1. *Given a Bipartite Graph, $G = (T \cup S, E)$, a vertex set $S' \subseteq S$ is defined to be the Discriminating Code Set of G, if $\forall t \in T, N(t) \cap S'$ is unique, where $N(t)$ denotes the neighborhood of t. The* Minimum Discriminating Code Set *(MDCS) problem is to find the Discriminating Code Set of minimum size.*

Figure 8.2 represents the bipartite graph obtained from Figure 8.1, with 5 nodes in T, representing the 5 HVTs, and 40 nodes in S. An MDCS solution $S' \subseteq S$ of this graph consists of three nodes (indicated by the three colored nodes) which ensure that they provide a unique code to identify each of the 5 nodes in T (colors above the nodes of T indicate the unique combination of sensors activated).

Differential Immunity

Note that when a single attack can cripple all the defense configurations $\in C$, MTD cannot aid in improving the robustness. In Chapter 7, we introduced the notion of *differential immunity* that aims at measuring the amount of diversity between configurations $\in C$. In this work, we seek to design a configuration set C that is differentially immune, i.e $\delta = 1$. Each attack, allowed by the threat model defined later, can only cripple one defense configuration. This ensures maximum diversity of C and implies the highest robustness gains for the formulated MTD.

To design such a C for our MTD system, we first need to find multiple MDCS sets of a bipartite graph. For this purpose, we desire K differentially immune MDCS, abbreviated as K-δMDCS, where no two MDCS solutions share a common sensor placement point.

Definition 2. ***(K-δMDCS)*** *Given a Bipartite Graph, $G = (T \cup S, E)$, K vertex sets $S_i \subseteq S, i \in \{1, \ldots, K\}$ are defined to be K-δMDCS of G, if the following conditions hold– (1) all the sets S_i are MDCSs of graph G and (2) for all possible pairs of sets (S_i, S_j), $S_i \cap S_j = \emptyset$.*

First, we want to activate the minimum number of sensors placed in the network at any point in time. Hence, we use K sets, all of which are MDCS, i.e. have the smallest cardinality. Second, the use of differentially immune MDCS tries to optimize for robustness in adversarial settings. If an attacker were to attack a particular sensor placement point $s \in S$, it can hope to, at best, cripple a single MDCS $S_i \in C$ from uniquely fingerprinting HVT failure. If the defender selects an different MDCS, i.e. $S_j \in C$ $(j \neq i)$, then the attacker will not succeed in effecting the functionality of

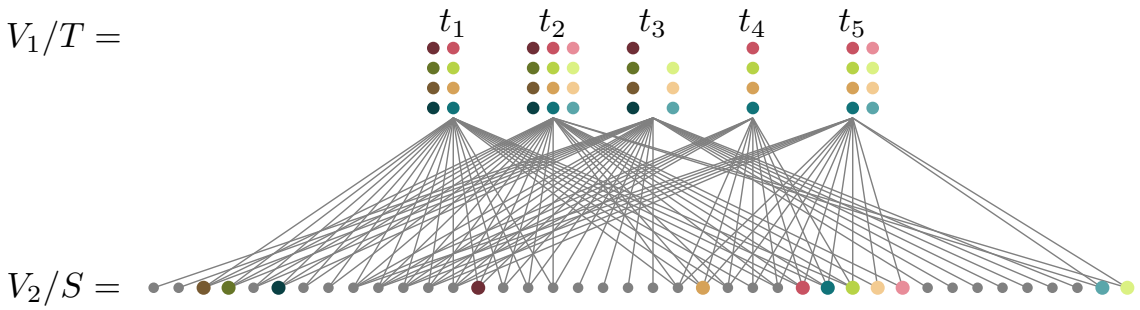


Figure 8.3: The IEEE 14-bus power grid graph has $4 - \delta$MCDS solutions.

the power grid sensors. We will now show that the decision problem corresponding to K-δMDCS is NP-complete.

Lemma 8.2.1. *K-δMDCS is NP-Complete, given K is an integer and $K > 0$.*

Proof. We note that the original MDCS problem, which is known to be NP-Complete [201], is simply a special case (when $K = 1$). □

Corollary 8.2.1.1. *K-δ Graph Problems such as K-δMinimum Identifying Code Set (MICS), K-δMinimum Set Cover (MSC), K-δMinimum Vertex Cover (MVC) are NP-Complete when K is an integer and $K > 0$.* [1]

Let us denote the size of an MDCS for a bipartite graph G as m. In K-δMDCS, the goal of the defender is to find K MDCSs each of size m. Then, the defender needs to place $K * m$ sensors in the power grid and, at any point in time, activate an MDCS set (of size m) to uniquely identify failures in T. While a large number of defender strategies (i.e. larger values of K) helps to increase their options for sensor activation in turn reducing the success rate for the attacker, it also incurs the cost

[1] Note that in the context of these problems, the distinction between the node sets T and S in MDCS are unnecessary and one can view the graphs as $G = (V, E)$.

of placing $K * m$ sensors. Thus, the ideal choice of K should trade-off robustness *vs.* sensor costs (when $K = 1$, robustness using MTD is impossible to achieve).

In cases where the defender has sufficient resources, one might ask *what is the maximum size of K?* Depending on the structure of the underlying graph, this question may have a trivial answer. For example, if the bipartite graph has a $t \in T$ and only a single $s \in S$ in its neighbourhood $N(t) = \{s\}$, any MDCS of G needs to place a sensor on s to detect a fault in t. Hence, there cannot exist two MDCSs that do not share a common node because s has to be a part of both. In such cases, the max value of K, denoted as $K_{\max}$, is 1. Beyond these special cases, similar to the problem of finding the maximum value of K in the K-clique problem, finding $K_{\max}$ demands a search procedure over the search space of K that we now describe.

Finding $K_{\max}$-δMDCS

We first propose a Quadratically Constrained Integer Linear Program (QCILP) that given a value of K, finds K Discriminating Code Sets (DCSs). We then showcase the algorithm for searching over possible values of $k \in \{1, \ldots, |S|\}$ to find the largest K. To define the QCILP for $G = (T \cup S, E)$, we first consider $|S| * k$ binary variables where $z_{sk} = 1$ if a sensor is placed in node $s \in S$ for the kth DSC and 0 otherwise. We also use a variable l that denotes the size of the DCSs found. We can now describe our QCILP as follows.

$$\min_{l,z} \quad l \tag{8.1}$$

$$s.t. \quad l = \sum_{s} z_{sk} \quad \forall k \qquad \text{All } k \text{ DCS has the same size } l.$$

$$\sum_{s \in S} (z_{sk} - z_{sk'})^2 = 2l \quad \forall (k, k') \qquad \text{No two DCSs should have a common sensor.}$$

$$\sum_{s \in N(t)} z_{sk} \geq 1 \quad \forall t, \forall k \qquad \text{All } t \in T \text{ has a sensor monitoring them for all the } k \text{ solutions.}$$

$$\sum_{s \in N(t) \Delta N(t')} z_{sk} \geq 1 \quad \forall (t, t'), \forall k \quad t \text{ and } t' \text{ trigger unique sensors for the } k\text{-th DCS.}$$
$$z_{sk} \in 0, 1 \forall s, \forall k$$

The last two constraints ensure that each of the K solutions is a Discrimination Code Set where (1) all $t \in T$ trigger at least one sensor $s \in S$ and (2) for all pairs of t and t' (both $\in T$), there exists at least one sensor in the symmetric difference set of t and t' that is a part of the DCS, which in turn uniquely distinguishes between t and t'. The first two constraints ensure that all k DCSs are of equal size and no two DCSs shares a common sensor. We can now ask the question as to whether the DCSs found by Equation 8.1 is indeed the Minimum DCSs (MDCSs) for the graph G. In this regard, we now show the following.

Theorem 8.2.2. *For all values $K \leq K_{\max}$, the optimization problem in Equation 8.1 returns K-δMDCS.*

Proof. We consider proof by contradiction. Given the value of $K(\leq K_{\max})$, let us assume that the solution returned by Equation 8.1 is not the K-δMDCS for the graph G. If this is the case, at least one of the two properties in the definition K-δMDCS is violated. Thus, either (1) the returned solution consists of DCS that is not the Minimum DCS, or (2) there exists a sub-set (of size greater than one) among the set of DCSs that share a common node.

Owing to the third and fourth constraints, all the solutions constitute a DCS. Now, if (1) is violated, all the DCSs returned by the QCILP, of length l, are not the MDCS for G. Thus, the MDCS must have a DCS of size $l' \leq l$. Given that the minimization objective finds the smallest DCS and $K \leq K_{\max}$, this cannot be possible. Hence, (1) does not hold.

For (2), let us say that there exists a subset of the DCSs returned that share a common node. If this was the case, then at least one solution pair has to share a common node. If this node is denoted as s^* and the two solutions are termed as k and k', then for the second constraint, given $z_{s^*k} = z_{s^*k'} = 1$, the term for s^* is zero. Even if the other $l-1$ nodes in the solutions k and k' are unique, the terms will add up to $2 * (l-1)$ thereby violating the second constraint. This is not possible and as a consequence, (2) does not hold. □

Given this, we can now consider cases where $K > K_{\max}$. When $K > K_{\max}$, the optimization problem in Equation 8.1 is either infeasible or returns K DCSs that are not MDCS for graph G. This condition holds by the definition of $K_{\max}$ (proof by contradiction ensues neither of the two cases holds). With these conditions in mind we can design an iterative approach, shown in Algorithm 5, to find the $K_{\max}-\delta$MDCS of a given graph.

Figure 8.3 showcase the $4-\delta$MDCS solutions returned by Algorithm 5 for the 14-bus power grid network. The different colors indicate the different MDCSs found for G and the shades of the same color indicate an MDCS set. As shown, each of the four MDCS has a size of $l = 3$ and uniquely identifies all the transformers T. The lack of overlapping colors in the bottom set of nodes indicates that no two MDCS share a common $s \in S$.

While the procedure in Algorithm 5 finds the $K_{\max} - \delta$MDCS, it can be time-consuming for the largest networks (although it works well on large power-grids as shown in the experimental section). Thus, one can consider a greedy approach in which one solves the MDCS problem using using an iterative ILP appraoch proposed in [194]. In the greedy algorithm, we solve the ILP for finding MDCS with the additional constraints that $z_s = 0$ for all the sensors found in the previous solution

Algorithm 5 Finding $K_{\max} - \delta$MDCS.

In: $G = (T \cup S, E)$

Out: $K_{\max} - \delta$MDCS

solutions $\leftarrow \emptyset$

$K \leftarrow 1$

while do$K \leq |S|$

    solutions$_K \leftarrow$ Solve Equation 8.1 with K

    if solutions$_K == \emptyset$ **then**

        break Infeasible for $K > K_{\max}$

    end if

    if solutions $! = \emptyset$ & $|\text{solutions}(l)| < |\text{solutions}_K(l)|$ **then**

        break DCS returned is not MDCS for $K > K_{\max}$

    end if

    solutions $\leftarrow$ solutions$_K$

    $K \leftarrow K + 1$

end while

return solutions

steps and continue doing so until (1) the ILP becomes infeasible or (2) results in DCS that does not have minimum cardinality. In the experimental section, we will see that although this approach is faster, it can output K-δMDCS where $K < K_{\max}$. The sub-optimality is a result of the constraint ordering "enforced" on the sensors that makes the ILP infeasible in the latter iterations.

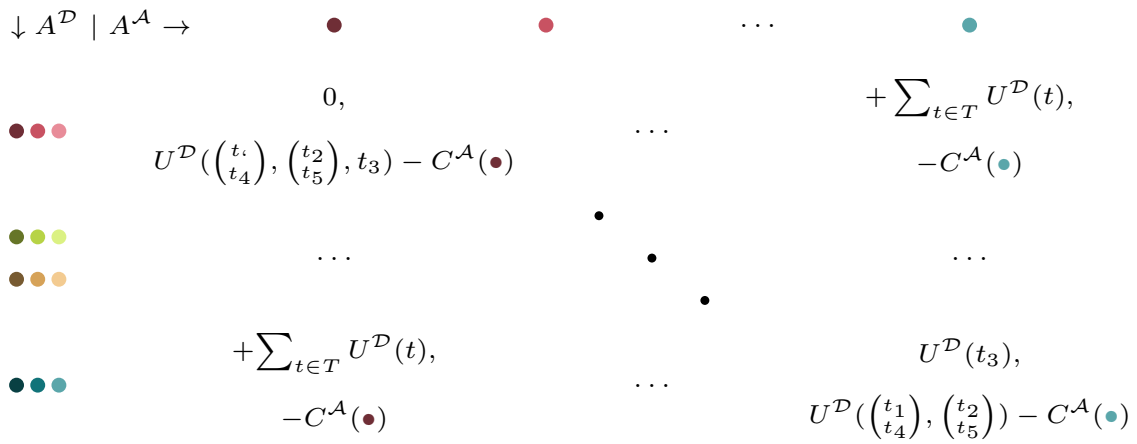


Figure 8.4: Game-matrix for the dynamic sensor activation problem.

8.3 Game Theoretic Formulation

The defender's goal is to maintain the unique identifying capability of HVTs at all times. Conversely, the attacker tries to prevent this capability, thereby making it harder for the defender to uniquely fingerprint impending failure of HVTs. In this section, we seek to find an optimal movement function M for the sensor activation MTD to aid the defender to realize its objective. To do so, we consider a strong threat-model where the attacker $\mathcal{A}$ with reconnaissance, is aware of the defender $\mathcal{D}$'s (probabilistic) sensor activation strategy, thereby making the Stackelberg Equilibrium an appropriate solution concept for our setting. We use a polynomial-time approach to find the Strong Stackelberg Equilibrium of the game [62]. We now briefly describe the various parameters of the formulated game (see Figure 8.4).

Defense Actions The defender has $K_{\max}$ pure strategies and the configuration set $C = K_{\max} - \delta$MDCS. If one uses the greedy algorithm instead of the optimal approach (both described in the previous section), the number of pure strategies obtained may be less than $K_{\max}$.

Attack Actions We assume that an attacker can spend reconnaissance effort in figuring out the sensor placement point. Thus, its action set includes attacking a sensor that may be considered for activation (instead of all the nodes in $|S|$). While one can consider attackers with the capability to attack multiple sensor activation points, it is often too expensive a cost model as it demands resource procurement and distribution over a wide geographic area.

Player Utilities The game has two different kinds of utilities that are used to calculate the rewards. First, the defender receives the utility associated with uniquely identifying a transformer $t \in T$ in the case of anomalous spikes indicative of failure (to occur). We assume that a transformer supplying power to an important building (eg. the White House or the Pentagon) is considered to be more important than one supplying power to a residential area. Second, the attacker's cost for attacking a particular sensor needs to be considered. While some sensors may be placed in high-security areas, others may be easier to access. We conduct randomized trails with both these values $\in [0, 10]$, with 10 indicating the HVT/sensor most important to protect/difficult to attack.

In the bottom right corner of Figure 8.4, the defender, owing to the attacker attacking a sensor, is only able to uniquely identify t_3 and thus, only gets reward proportional to it. Contrarily, the attacker, due to attacking a sensor, can make failures of t_1 and t_2 (and t_4 and t_5) indistinguishable and receives the corresponding utilities, minus the cost of attacking the sensor denoted by the light blue node ($\in S$, Figure 8.3). Similarly, if the attacker selects the attack represented by the first attack column (sensor denoted by the dark brown node), the defender cannot identify any HVT and thus, gets a utility of zero.

Graph	$\|S\|+\|T\|$	C		Movement Function M			
		$\|A^{\mathcal{D}}\|$ ($K/K_{\max}$)	$\|A^{\mathcal{A}}\|$ ($K/K_{\max}$)	URS (K)	URS ($K_{\max}$)	SSE (K)	SSE ($K_{\max}$)
14 Bus	45	4/4	12/12	18.5±4.7	18.65±4.7	20.62±4.6	**20.72**±4.6
30 Bus	89	4/4	16/16	26.45±5.7	27.25±5.6	29.44±6	**29.9**±5.8
39 Bus	96	7/9	28/36	18.7±5	19.24±5.2	**19.8**±5.3	19.73±5.3
57 Bus	170	6/6	60/60	70.76±10.8	70.88±11.1	**73.5**±10.6	73.07±10.7
89 Bus	422	16/21	96/126	50.67±8.9	51±9	**52.2**±9.2	**52.2**±9.2
118 Bus	367	2/2	10/10	31.35 ±6	31.6 ± 6	32.45±6.4	**32.61**±6.1
2383 Bus	5927	2/3	212/318	832.7±38.7	836.16±36.7	835.34±39	**842.34**±39.4

Table 8.1: Game parameters and defender's reward for playing the different Cs and Ms for the various power-grid networks.

8.4 Experimental Simulations

In this section, we conduct simulation studies on seven IEEE test graphs popular in the power domain [202]. The total number of nodes (i.e. $|S| + |T|$) for each test case is shown in Table 8.1. The table further lists the K values for the K-δMDCS found by the greedy (denoted by K) and the optimal Algorithm 5 (denoted by K_{max}). The size of the attacker's pure-strategy set is listed in the fourth column; this value can be obtained by multiplying the corresponding K value with the size of an MDCS for graph G as none of the K-δMDCS share a common node. We now present two results– (1) the effectiveness of the game-theoretic equilibrium compared to the Uniform Random Strategy baseline (which chooses to activate a particular MDCS with equal probability) and (2) the solution quality of alongside the time taken by the greedy and the optimal algorithms.

Effectiveness of Game-Theoretic Equilibrium

In Table 8.1, we show that in all test cases, the optimal movement strategy at the Strong Stackelberg Equilibrium (SSE) gives the defender a higher reward than choosing URS. When using URS or SSE, in most cases we see higher gains when the construction of the MTD configuration set C is optimal ($URS(K_{\max})$ obtained from Algorithm 5) as opposed to using a greedy algorithm ($URS(K)$). We expected this as the higher number of differentially immune options (as $K_{\max} > K$) chosen with equal probability reduces the probability of picking the weakest strategy. When the value of $K_{\max} = K$, such as for 14, 30, 57 and 118 buses, we see that the difference between the two versions of URS (or two versions of SSE) are negligible. The non-zero difference between the rewards values arises because of the particular MDCS sets chosen even though the total number of sets chosen are the same. We also see that the difference in the defender's utility can be large even when the difference between K and $K_{\max}$ is small in the case of larger networks (eg. 2383 bus). Thus, without finding the $K_{\max}$ and the SSE for the optimal C, it is hard to the establish the loss in rewards. Given that these strategies are pre-computed, the power grid utility operator should not consider the greedy strategy unless the time required by Algorithm 5 becomes prohibitive, as aspect we will now discuss.

Computational Time for finding C

In Figure 8.5, we compare the time taken for finding the configuration set C using the optimal *vs.* the greedy approach. We choose the logarithmic scale for the y-axis because the computational time of the optimal and greedy approaches for the 14, 30, 39, 57, and 118 buses was less than a second, and thus difficult to distinguish between on a linear scale. The largest disparity occurs when the size of the optimal

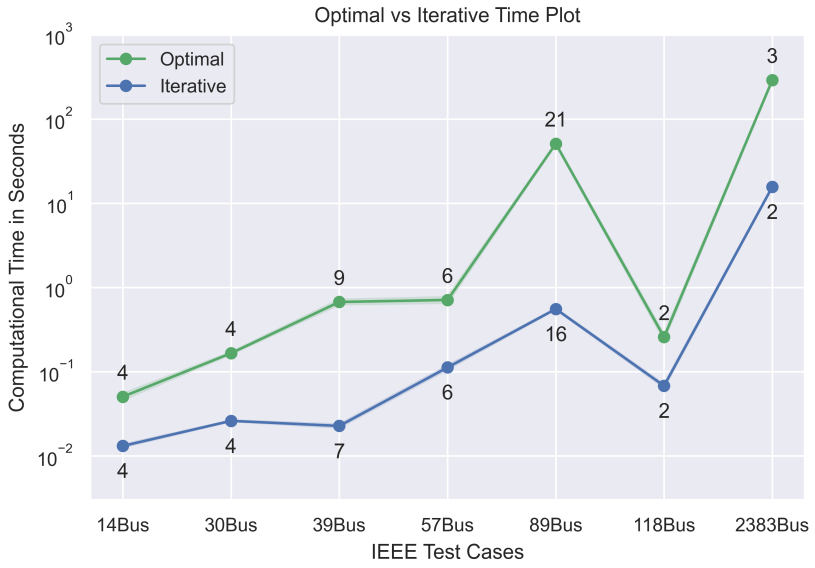


Figure 8.5: Time taken by the optimal (Algorithm 5) *vs.* the greedy approach for finding $K_{\max}-\delta$MDCS and K-δMDCS (the K values are shown above the plot points).

set $K_{\max}$ is greater than the K-sized set found by the greedy approach (39/89/2383 Bus). In other cases, while the optimal approach is slower, it provides the guarantee that no solution with a greater K exists, which is absent in the greedy case. A case where the logarithmic scale, from a visualization perspective, does not do justice is the 2383-Bus. The time taken by the greedy approach is $15s$ compared to $291s$ taken by the optimal approach. While the K value differs by a factor of one, the resultant gain in defender's game value, as shown in Table 8.1, is relatively large. Thus, the added time in generating the optimal configuration set needs to be criticized based on the gain obtained in the underlying game.

We also consider the pragmatic scenario when the K value is fixed by the defender up-front owing to budget restrictions of sensors that can be placed in the power network. In this case, the greedy approach has to iteratively find one solution at a time, adding them to the constraint set of future iterations until the desired k is

reached. On the other hand, the iterative procedure in Algorithm 5 can be altogether ignored and one can simply return the solution found by the optimization problem in Equation 8.1.

8.5 Related Works

Adversarial attacks on power grids comprise of false-data injection, jamming, DoS and packet-dropping attacks [203, 204, 198]. While researchers have proposed a multitude of defense mechanisms [205], including Moving Target Defense (MTDs) [206, 207], they do not consider the problem of sensor placement to monitor HVTs. On the other hand, works that leverage the formalism of Discriminating Code Sets [201] to optimize sensor placement [194], have focused on scalability issues and provided theoretical bounds in these settings [208]; completely ignore the issue of robustness to adversarial intent. In this work, we attempted to fill in this gap.

While an array of research work has formally investigated the notion of finding an optimal movement function M for MTDs, the configuration set C is pre-decided based on heuristic guidance from security experts [209]. While some works consider the aspect of differential immunity by analyzing code overlap for cyber systems [22] or Jacobians of gradients for deep neural networks [180], these measures have no way of ensuring differential immunity. The notion of k-set diverse solutions in Constraint Satisfaction Programming (CSP) [210], although conceptually similar to our notion of differential immunity, does not have the added constraint of finding a minimum sized solution (as in the case of MDCS). In adversarial scenarios, our work is the first to formalize the notion of diversity in graphs and propose linear programming methods to find them.

8.6 Concluding Remarks

In this chapter, we considered the problem of monitoring the behavior of HVTs in adversarial settings. We suggested an approach based on MTD and formulated it as a game between the power utility company (the defender) and an adversary. We showed that finding a differentially immune configuration set for the defender is NP-Complete and presented two algorithms– an optimal QC-ILP and a greedy iterative-ILP. Movement strategies at Stackelberg Equilibrium characterize optimal behavior in the face of adversarial attacks and enable a defender to activate a limited number of sensors at any point in time while being able to uniquely identify failure points. Results obtained on several IEEE test cases show that the optimal configuration construction coupled with the optimal movement strategy provides the defender the highest gain in rewards.

Chapter 9

CONCLUSION

Change is the only constant.

– Heraclitus

Conventional wisdom dictates that a moving system is less predictable and therefore, more difficult to attack. This wisdom, however, only holds when movement is based on careful analysis of existing data and considers strategic behavior by an adversary. Thus, reasoning about the *what, when,* and *how* of movement becomes a critical aspect of leveraging movement for security. In this concluding chapter, we discuss three aspects. First, we re-state the goals of our research and discuss how the works presented in this thesis help achieve them. Second, we reflect on the chosen methodology and emphasize on the novel questions that arise, highlighting directions for future work. Finally, we highlight the key takeaways for readers who seek to use Moving Target Defenses.

9.1 Aims of this Thesis

As stated in the introduction, the thesis has three aims. In this section, we describe each of them and highlight how the research presented in the thesis achieves them.

First, we noted that conventional Moving Target Defenses (MTDs) consider naïve movement strategies, leaving open the problem of designing intelligent movement strategies. We hypothesized that when domain knowledge is available, we can model these problems as a multi-agent interaction and proposed methods to infer movement

strategies that improve the effectiveness of MTDs. In Chapter 3 and 4, we showed that by modeling MTD as a two-person game, leveraging existing domain knowledge and devising methods to infer strategies at Strong Stackelberg Equilibrium, we can significantly outperform existing movement strategies. Second, when domain knowledge is not readily available, but interaction with a system is possible, we asked whether we can characterize and learn optimal movement strategies. In Chapter 5, we proposed a unified game-theoretic framework that can characterize optimal movement strategies for both the web-application security and the cloud network security domains. Then, we developed a novel variant of multi-agent Reinforcement Learning and showed the learned movement strategy converges to the optimal movement strategy.

Third, we investigated if the idea of Moving Target Defenses can be readily applied to novel application domains and, in doing so, can it be more effective than current defense mechanisms in these settings? In Chapter 6, we married the two most popular proactive defense mechanisms in cyber-security and proposed a moving target approach to cyber-deception in software security. We were able to show that static deception techniques eventually leak information to an attacker about which vulnerabilities are honey-patches. Using MTD, the attacker is not only kept guessing but upon failure helps the defender to counter-surveil them and gather valuable information.

In Chapter 7, we consider moving the classification surface of Deep Neural Networks which acts as (1) an attack surface for white-box attacks and (2) an exploration surface for model theft and black-box attacks. The lack of differential immunity becomes a hindrance to the success of MTD in this scenario. Yet, with a softer notion of differential immunity inherently possessed by an ensemble of existing Deep Neural Networks, we show that the MTDeep acts as an add-on defense-in-depth strategy

that pushes the robustness of classifiers beyond state-of-the-art on image classification tasks. Further, owing to the in-built randomness at classification time, black-box attacks, created using approaches similar to model theft, becomes less potent.

In Chapter 8, we proposed an MTD for shifting the detection surface responsible for fingerprinting failures in power networks. By leveraging graph-theoretic modeling of these networks, we formally defined the problem of finding differentially immune sensor placements and designed optimization problems to find them. Beyond the concept of finding a Minimum Discriminating Code Set (MDCS), novel problems emerged when we considered the notion of differential immunity to other graph-theoretic solution concepts. We then leveraged our game-theoretic formalism to find optimal movement strategies in this setting and showed that it provides robustness to adversarial attacks.

9.2 Reflections and Future Work

The work presented in this thesis considers the use of multi-agent systems in the context of Moving Target Defense (MTD) for cyber-security followed by the use of MTD in novel application domains, spanning from cyber-physical system to machine learning. While we mostly focus on developing a unified multi-agent formalism for MTD that lets us model properties of the configuration set C and development of optimal movement strategies M, progress on several fronts is necessary to enable the effective use of MTD in all the domains presented in the thesis. This section takes a critical view of these defenses, in places highlighting relevant work and suggesting future directions.

Accurate modeling of cyber-interaction

A true cyber-interaction is often not limited to the inherent limitations of various game-theoretic and graph-theoretic frameworks discussed in the thesis. For example, the assumption about complete observability may be too strong a threat model in the context of Chapter 3 and 6. We not only assumed that the attacker is aware of the vulnerabilities in the code-base, but also considered they know the patches used and the movement strategy. On the other hand, assuming that the defender can fully observe an attacker's movement in Chapter 4 constitutes a weak threat model. To justify this assumption, we showed in Chapter 5 that consideration of partial observability makes (1) inference time-consuming and (2) learning sample-inefficient. Even in single-agent modeling, partial observability makes it difficult for strategy inference mechanisms to scale beyond 30 nodes [51]. While the use of these models for improving the effectiveness of MTD is important, trading-off the pragmatism afforded by a model *vs.* the scalability of inference (or the sample-complexity of learning) is an important question.

Development and Maintenance of MTDs

With the complexity of modern cyber-systems, it is often a formidable task to even design a single software or network that achieves all the user requirements. In this regard, the requirement of having a set of fully functional system configurations, necessary for MTD, may seem unreasonable in some contexts. While the ever-evolving space of attacks is expected to keep all defense systems on their toe, we can categorize the work presented in this thesis into two buckets based on the relative costs of deployment and maintenance.

High Development costs, High Maintenance Costs In Chapter 3, we presented an approach to improving the effectiveness of MTDs in web-applications. In this setting, a practitioner has to develop multiple code-bases that support the web-application logic, deal with movement costs [20], and maintenance issues. While we try to account for switching costs in Chapter 3, the development of automatic code translation mechanisms [69] are equally important to facilitate the use of MTD in these settings. Similarly, in Chapter 8, for the dynamic sensor placement in power networks, one needs more Power Monitoring Units (PMUs) than what would be used at any given point in time. This drives up both the development costs and the maintenance costs for power networks. Fortunately, the cost of coming up with several placement points for MTD remains similar to finding a single-placement point for static placement.

High Development Costs, Low Maintenance Costs In our development of MTD for IDS system placement discussed in Chapter 4, one incurs high development costs of setting up a centralized SDN system that can facilitate movement and inferring movement strategies. Once it is deployed, the maintenance costs are relatively less. A similar line of reasoning exists for our works on moving target cyber-deception in Chapter 6 and MTDeep in Chapter 7. In the former, one needs significant development effort to collect or develop regular and honey-patches that can be used by the software system. Similarly, in the latter case, training large neural network models that give high accuracy on the task at hand (and differential immunity) may incur high development costs. In some cases, the development costs is a question of ongoing research and thus, substantially large. In comparison, maintenance of these systems, once developed, is far less.

By no means is the categorization of works, in this space of development and maintenance costs, exhaustive. While all the MTDs discussed in our survey [7] can be situated in this categorization, given the context of this thesis, we simply consider two regions in this space.

Asymmetric Performance of Different System Configurations

Often, the constituent configurations of an MTD are not equally performant, thereby resulting in asymmetric performance impacts. In Chapter 3, while we address the performance impact of switching costs for web-application MTD, we do not let the quality of service (QoS) metric associated with a single configuration (eg. latency of response) affect the rewards associated with deploying a particular system. On the other hand, we encode the QoS metrics in the utilities of our formulated game in case of dynamic IDS placement in Chapter 4 and moving target patch deployment in Chapter 6. Further, the rewards returned by the environment lets us encode this impact in the strategy learning scenario discussed in Chapter 5. Although consideration of the performance impact in the rewards of the formulated game lets us infer or learn strategies that consider both security and performance impacts, there are other ways to model QoS costs. In Chapter 7, we separate out the QoS metric (accuracy on the test data) and the robustness metric (accuracy on adversarially perturbed input) into two players and then use a parameter to trade-off the importance given to accuracy *vs.* robustness, making it a Bayesian normal-form game.

In some MTD scenarios, this concern ceases to exist. For example, in Chapter 8, we explicitly ensure that the number of PMUs activated in the power system is optimal and all of them are capable of uniquely identifying failures in High Voltage

Transformers (HVTs). Hence, one can expect the power requirements of the constituent MTD configurations to be similar.

Implementing Movement

A static software system is completely oblivious to the engineering effort required for and the performance impacts of MTDs. Shifting from a particular configuration c to another c' can greatly vary depending on the particular MTD under consideration. In the context of web-applications, we try to explicitly account for this cost when inferring a movement strategy. In Chapter 5, we expect the environment to execute a switch action and, depending on the operational metrics observed (encoded as stochasticity), successfully switch or disregard the movement by falling back to the existing configuration. In dynamic IDS placement, the availability of a centralized controller in a Software-Defined Network simplifies movement to a script that shuts down and brings up multiple intrusion detection processes (across the system). While we consider emulation cloud scenarios in Chapter 4, we categorize several MTDs in network security based on the maturity of experiments, ranging from experiments on simulation environments to industrial test-beds, in our survey [209].

On the other hand, the use of MTD in novel application domains makes it harder to leverage existing technologies to facilitate movement unless it is trivial. In Chapter 7, MTDeep simply has to reroute input images to constituent classifiers using a pre-computed movement strategy whereas we had to design an end-to-end system architecture and dynamic patch-set selection mechanism for moving target cyber-deception in Chapter 6. On similar lines, for robust sensor activation in power networks in Chapter 8, we might need to consider the effort required at the ground level to activate and deactivate Power Monitoring Units (PMUs).

Prioritization for Patching Known Vulnerabilities

Given the complexity of current software systems, discovery and patching of existing vulnerabilities is commonplace. With the large number of vulnerabilities discovered each year ($\approx$ 9351 on average in the last 8 years [1]), the hope of fixing all vulnerabilities at once is a distant dream. Hence, defenders must prioritize the limited effort that can be devoted to patch vulnerabilities. In Chapter 3 and 4, we show that MTD systems exacerbate this problem. Not only does the use of multiple configurations increase the count of known vulnerabilities, the additional knowledge used to model the multi-agent interaction makes vulnerability prioritization more difficult. While we suggest brute-force approaches to identify them, the development of better algorithms is important for the future.

Introduction of Novel Security Concerns

The use of Moving Target Defense opens up a Pandora's box of novel vulnerabilities. We will discuss a few of them here.

Attack Surface Expansion While the use of MTD limits an attacker's success in terms of determining the configuration of a system at the time of attack, the use of multiple configurations broadens the overall attack surface and forces the defender to consider a larger set of vulnerabilities. The larger attack surface, in turn, exacerbates the problem of routine maintenance such as monitoring exploits and patching existing vulnerabilities. In the context of multi-stage attacks such as Advanced Persistent Threats (APTs) discussed in Chapter 4, a stealthy attacker who resides in one of the nodes may be favored to succeed at times. For example, given a static system,

[1] Estimate obtained by averaging data gathered form https://www.cvedetails.com.

the attacker's position within a cloud network may be of a disadvantage if the nodes reachable from this vantage point all have vulnerabilities that are beyond the expertise level of an attacker. In the context of an MTD, the attacker can hope that the configurations of the accessible systems will eventually change over time making them vulnerable to an attack in the attacker's arsenal.

Centralized Implementation Techniques In Chapter 4, MTD techniques necessitate the use of a centralized mechanism such as Software Defined Networking (SDN) to send instructions to firewalls, hosts, etc. While strong measures can be taken to ensure that the centralized controller is well protected against adversarial attacks, one cannot guarantee its security, especially in light of zero-day vulnerabilities, creating a single point of failure. Hence, some of the technologies that are essential to enable the use of MTD in real-world scenarios also add to the security challenges of the defender.

9.3 Takeaways

The three key takeaways relate closely to the goals of this thesis. First, consideration of MTD in adversarial environments can greatly improve the security guarantees provided by existing security mechanisms in several domains. Being a defense-in-depth solution, MTD can be used in conjunction with the existing defense mechanism. This guarantees that using MTD can never be worse in terms of security. Second, when existing knowledge is available about the adversary, the threat model, and the performance impact of movement, modeling MTD as a game can help in characterizing and inferring optimal movement strategies that strike a balance between security and performance. Lastly, when information about system dynamics and the effect of actions on a system is unknown but interaction with the MTD system is possi-

ble, one can simply learn the optimal movement strategies by adapting techniques in Multi-agent Reinforcement Learning.

REFERENCES

[1] Rui Zhuang, Scott A DeLoach, and Xinming Ou. Towards a theory of moving target defense. In *Proceedings of the First ACM Workshop on Moving Target Defense*, pages 31–40. ACM, 2014.

[2] Richard Klima, Karl Tuyls, and Frans Oliehoek. Markov security games: Learning in spatial security problems. In *NIPS Workshop on Learning, Inference and Control of Multi-Agent Systems (2016)*, pages 1–8, 2016.

[3] Kasey Panetta. Gartner's Top 10 Security Predictions 2016. https://www.gartner.com/smarterwithgartner/top-10-security-predictions-2016/, 2016. Online; accessed 11 Nov 2018.

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

[5] Sushil Jajodia, Anup K Ghosh, Vipin Swarup, Cliff Wang, and X Sean Wang. *Moving target defense: creating asymmetric uncertainty for cyber threats*, volume 54. Springer Science & Business Media, 2011.

[6] 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.

[7] Sailik Sengupta, Ankur Chowdhary, Abdulhakim Sabur, Dijiang Huang, Adel Alshamrani, and Subbarao Kambhampati. A survey of moving target defenses for network security. *IEEE Communications Surveys & Tutorials*, 2020.

[8] Ehab Al-Shaer, Qi Duan, and Jafar Haadi Jafarian. Random host mutation for moving target defense. In *International Conference on Security and Privacy in Communication Systems*, pages 310–327. Springer, 2012.

[9] Massimiliano Albanese, Alessandra De Benedictis, Sushil Jajodia, and Kun Sun. A moving target defense mechanism for manets based on identity virtualization. In *2013 IEEE Conference on Communications and Network Security (CNS)*, pages 278–286. IEEE, 2013.

[10] Jafar Haadi Jafarian, Ehab Al-Shaer, and Qi Duan. Openflow random host mutation: transparent moving target defense using software defined networking. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 127–132. ACM, 2012.

[11] Abdullah Aydeger, Nico Saputro, Kemal Akkaya, and Mohammed Rahman. Mitigating crossfire attacks using sdn-based moving target defense. In *Local Computer Networks (LCN), 2016 IEEE 41st Conference on*, pages 627–630. IEEE, 2016.

[12] Zheng Zhao, Fenlin Liu, and Daofu Gong. An sdn-based fingerprint hopping method to prevent fingerprinting attacks. *Security and Communication Networks*, 2017, 2017.

[13] 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*, pages 892–900. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[14] Sushil Jajodia, Noseong Park, Edoardo Serra, and VS Subrahmanian. Share: A stackelberg honey-based adversarial reasoning engine. *ACM Transactions on Internet Technology (TOIT)*, 18(3):30, 2018.

[15] Ramazan Algin, Huseyin O Tan, and Kemal Akkaya. Mitigating selective jamming attacks in smart meter data collection using moving target defense. In *Proceedings of the 13th ACM Symposium on QoS and Security for Wireless and Mobile Networks*, pages 1–8. ACM, 2017.

[16] Cheng Lei, Duo-He Ma, and Hong-Qi Zhang. Optimal strategy selection for moving target defense based on markov game. *IEEE Access*, 5:156–169, 2017.

[17] Sailik Sengupta, Tathagata Chakraborti, and Subbarao Kambhampati. Mtdeep: Moving target defense to boost the security of deep neural nets against adversarial attacks. *International Conference on Decision and Game Theory for Security*, 2019.

[18] Pratyusa K Manadhata. Game theoretic approaches to attack surface shifting. In *Moving Target Defense II*, pages 1–13. Springer, 2013.

[19] Quanyan Zhu and Tamer Başar. Game-theoretic approach to feedback-driven multi-stage moving target defense. In *International Conference on Decision and Game Theory for Security*, pages 246–263. Springer, 2013.

[20] Sailik Sengupta, Satya Gautam Vadlamudi, Subbarao Kambhampati, Adam Doupé, Ziming Zhao, Marthony Taguinod, and Gail-Joon Ahn. A game theoretic approach to strategy generation for moving target defense in web applications. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pages 178–186. International Foundation for Autonomous Agents and Multiagent Systems, 2017.

[21] Satya Gautam Vadlamudi, Sailik Sengupta, Marthony Taguinod, Ziming Zhao, Adam Doupé, Gail-Joon Ahn, and Subbarao Kambhampati. Moving target defense for web applications using bayesian stackelberg games. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 1377–1378. International Foundation for Autonomous Agents and Multiagent Systems, 2016.

[22] Kevin M Carter, James F Riordan, and Hamed Okhravi. A game theoretic approach to strategy determination for dynamic platform defenses. In *Proceedings of the First ACM Workshop on Moving Target Defense*, pages 21–30. ACM, 2014.

[23] Michael Thompson, Nathaniel Evans, and Victoria Kisekka. Multiple os rotational environment an implemented moving target defense. In *Resilient Control Systems (ISRCS), 2014 7th International Symposium on*, pages 1–6. IEEE, 2014.

[24] Ankur Chowdhary, Sandeep Pisharody, and Dijiang Huang. Sdn based scalable mtd solution in cloud network. In *Proceedings of the 2016 ACM Workshop on Moving Target Defense*, pages 27–36. ACM, 2016.

[25] Iman El Mir, Ankur Chowdhary, Dijiang Huang, Sandeep Pisharody, Dong Seong Kim, and Abdelkrim Haqiq. Software defined stochastic model for moving target defense. In *International Afro-European Conference for Industrial Advancement*, pages 188–197. Springer, 2016.

[26] Saptarshi Debroy, Prasad Calyam, Minh Nguyen, Allen Stage, and Vladimir Georgiev. Frequency-minimal moving target defense using software-defined networking. In *2016 International Conference on Computing, Networking and Communications (ICNC)*, pages 1–6. IEEE, 2016.

[27] Achintya Prakash and Michael P Wellman. Empirical game-theoretic analysis for moving target defense. In *Proceedings of the Second ACM Workshop on Moving Target Defense*, pages 57–65. ACM, 2015.

[28] Saran Neti, Anil Somayaji, and Michael E Locasto. Software diversity: Security, entropy and game theory.

[29] Michael Crouse, Errin W Fulp, and Daniel Canas. Improving the diversity defense of genetic algorithm-based moving target approaches. In *Proceedings of the National Symposium on Moving Target Research*, 2012.

[30] Bhakti Bohara. *Moving Target Defense Using Live Migration of Docker Containers*. PhD thesis, Arizona State University, 2017.

[31] Ankur Chowdhary, Sandeep Pisharody, Adel Alshamrani, and Dijiang Huang. Dynamic game based security framework in sdn-enabled cloud networking environments. In *Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, pages 53–58. ACM, 2017.

[32] Andrew Clark, Kun Sun, Linda Bushnell, and Radha Poovendran. A game-theoretic approach to ip address randomization in decoy-based cyber defense. In *International Conference on Decision and Game Theory for Security*, pages 3–21. Springer, 2015.

[33] Richard Colbaugh and Kristin Glass. Predictability-oriented defense against adaptive adversaries. In *Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on*, pages 2721–2727. IEEE, 2012.

[34] Sridhar Venkatesan, Massimiliano Albanese, George Cybenko, and Sushil Jajodia. A moving target defense approach to disrupting stealthy botnets. In *Proceedings of the 2016 ACM Workshop on Moving Target Defense*, pages 37–46. ACM, 2016.

[35] Sailik Sengupta, Ankur Chowdhary, Dijiang Huang, and Subbarao Kambhampati. Moving target defense for the placement of intrusion detection systems in the cloud. In *International Conference on Decision and Game Theory for Security*, pages 326–345. Springer, 2018.

[36] Sailik Sengupta, Ankur Chowdhary, Dijiang Huang, and Subbarao Kambhampati. General sum markov games for strategic detection of advanced persistent threats using moving target defense in cloud networks. 2019.

[37] Sailik Sengupta, Kaustav Basu, Arunabha Sen, and Subbarao Kambhampati. Moving target defense for robust monitoring of electric grid transformers in adversarial environments. In *Conference on Decision and Game Theory for Security (GameSec)*, 2020.

[38] Ankur Chowdhary*, Sailik Sengupta*, Dijiang Huang, and Subbarao Kambhampati. Markov game modeling of moving target defense for strategic detection of threats in cloud networks. *AAAI Workshop on Artificial Intelligence for Cyber Security (AICS)*, 2019.

[39] Frederico Araujo, Sailik Sengupta, Jiyong Jang, Adam Doupe, Kevin W. Hamlen, and Subbarao Kambhampati. Software deception steering through version emulation. In *Hawaii International Conference on System Sciences (HICSS)*, 2021.

[40] Sailik Sengupta and Subbarao Kambhampati. Multi-agent reinforcement learning in bayesian stackelberg markov games for adaptive moving target defense. *arXiv preprint arXiv:2007.10457*, 2020.

[41] Sailik Sengupta. Moving target defense: a symbiotic framework for ai & security. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pages 1861–1862. International Foundation for Autonomous Agents and Multiagent Systems, 2017.

[42] Sailik Sengupta, Ankur Chowdhary, Dijiang Huang, and Subbarao Kambhampati. General sum markov games for strategic detection of advanced persistent threats using moving target defense in cloud networks. In *International Conference on Decision and Game Theory for Security*, pages 492–512. Springer, 2019.

[43] Sailik Sengupta, Tathagata Chakraborti, and Subbarao Kambhampati. Mtdeep: boosting the security of deep neural nets against adversarial attacks with moving target defense. In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[44] Marten Van Dijk, Ari Juels, Alina Oprea, and Ronald L Rivest. Flipit: The game of "stealthy takeover". *Journal of Cryptology*, 26(4):655–713, 2013.

[45] Yuan Shi, Huanguo Zhang, Juan Wang, Feng Xiao, Jianwei Huang, Daochen Zha, Hongxin Hu, Fei Yan, and Bo Zhao. Chaos: An sdn-based moving target defense system. *Security and Communication Networks*, 2017, 2017.

[46] Iman El Mir, Ankur Chowdhary, Dijiang Huang, Sandeep Pisharody, Dong Seong Kim, and Abdelkrim Haqiq. Software defined stochastic model for moving target defense. In *International Afro-European Conference for Industrial Advancement*, pages 188–197. Springer, 2016.

[47] Henger Li, Wen Shen, and Zizhan Zheng. Spatial-temporal moving target defense: A markov stackelberg game model. *arXiv preprint arXiv:2002.10390*, 2020.

[48] Ankur Chowdhary, Adel Alshamrani, Dijiang Huang, and Hongbin Liang. Mtd analysis and evaluation framework in software defined network (mason). In *Proceedings of the 2018 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, pages 43–48. ACM, 2018.

[49] Ankur Chowdhary, Sailik Sengupta, Adel Alshamrani, Dijiang Huang, and Abdulhakim Sabur. Adaptive mtd security using markov game modeling. *arXiv preprint arXiv:1811.00651*, 2018.

[50] Hoda Maleki, Saeed Valizadeh, William Koch, Azer Bestavros, and Marten van Dijk. Markov modeling of moving target defense games. In *Proceedings of the 2016 ACM Workshop on Moving Target Defense*, pages 81–92. ACM, 2016.

[51] Erik Miehling, Mohammad Rasouli, and Demosthenis Teneketzis. Optimal defense policies for partially observable spreading processes on bayesian attack graphs. In *Proceedings of the Second ACM Workshop on Moving Target Defense*, pages 67–76. ACM, 2015.

[52] Thanh H Nguyen, Mason Wright, Michael P Wellman, and Satinder Singh. Multistage attack graph security games: Heuristic strategies, with empirical game-theoretic analysis. *Security and Communication Networks*, 2018, 2018.

[53] Quanyan Zhu and Tamer Başar. Dynamic policy-based ids configuration. In *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, pages 8600–8605. IEEE, 2009.

[54] Xiaofan He, Huaiyu Dai, Peng Ning, and Rudra Dutta. Dynamic ids configuration in the presence of intruder type uncertainty. In *2015 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2015.

[55] Lisa Oakley and Alina Oprea. Playing adaptively against stealthy opponents: A reinforcement learning strategy for the flipit security game. *arXiv preprint arXiv:1906.11938*, 2019.

[56] Taha Eghtesad, Yevgeniy Vorobeychik, and Aron Laszka. Deep reinforcement learning based adaptive moving target defense. *arXiv preprint arXiv:1911.11972*, 2019.

[57] BV Stengel and S Zamir. Leadership with commitment to mixed strategies. *CDAM Research Report LSE-CDAM-2004-01*, 2004.

[58] Heinrich Von Stackelberg and Stackelberg Heinrich Von. *The theory of the market economy*. Oxford University Press, 1952.

[59] Tamer Basar et al. Lecture notes on non-cooperative game theory. *Game Theory Module of the Graduate Program in Network Mathematics*, pages 3–6, 2010.

[60] George Leitmann. On generalized stackelberg strategies. *Journal of Optimization Theory and Applications*, 26(4):637–643, 1978.

[61] Praveen Paruchuri, Jonathan P Pearce, Janusz Marecki, Milind Tambe, Fernando Ordonez, and Sarit Kraus. Playing games for security: An efficient exact algorithm for solving bayesian stackelberg games. In *AAMAS, 2008*, pages 895–902, 2008.

[62] Vincent Conitzer and Tuomas Sandholm. Computing the optimal strategy to commit to. In *Proceedings of the 7th ACM conference on Electronic commerce*, pages 82–90. ACM, 2006.

[63] Sailik Sengupta, Zahra Zahedi, and Subbarao Kambhampati. To monitor or to trust: Observing robot's behavior based on a game-theoretic model of trust. *arXiv preprint arXiv:1903.00111*, 2019.

[64] Yevgeniy Vorobeychik and Satinder Singh. Computing stackelberg equilibria in discounted stochastic games (corrected version). 2012.

[65] Jessica Silver-Greenberg, Matthew Goldstein, and Nicole Perlroth. JPMorgan Chase Hacking Affects 76 Million Households. In *The New York Times*, 2014.

[66] Davide Balzarotti, Marco Cova, Vika Felmetsger, Nenad Jovanovic, Engin Kirda, Christopher Kruegel, and Giovanni Vigna. Saner: Composing static and dynamic analysis to validate sanitization in web applications. In *Security & Privacy 2008. IEEE Symposium*, pages 387–401, 2008.

[67] Adam Doupé, Ludovico Cavedon, Christopher Kruegel, and Giovanni Vigna. Enemy of the state: A state-aware black-box web vulnerability scanner. In *USENIX Security Symposium*, 2012.

[68] Dave Wichers. Owasp top-10 2013. *OWASP Foundation*, 2013.

[69] Marthony Taguinod, Adam Doupé, Ziming Zhao, and Gail-Joon Ahn. Toward a Moving Target Defense for Web Applications. In *IEEE Information Reuse and Integration (IRI)*, 2015.

[70] A. Sinha, T.H. Nguyen, D. Kar, M. Brown, M. Tambe, and A. X. Jiang. From physical security to cyber security. *Journal of Cybersecurity*, 2016.

[71] John McCumber. Information systems security: A comprehensive model. In *Proceedings of the 14th National Computer Security Conference*, 1991.

[72] Marthony Taguinod, Adam Doupé, Ziming Zhao, and Gail-Joon Ahn. Toward a Moving Target Defense for Web Applications. In *Proceedings of 16th IEEE IC-IRI*, 2015.

[73] Peter Mell, Karen Scarfone, and Sasha Romanosky. Cvss v2 complete documentation, 2007.

[74] Heinrich Von Stackelberg. *Market structure and equilibrium*. Springer SBM, 2010.

[75] Vincent Conitzer and Tuomas Sandholm. Computing the optimal strategy to commit to. In *Proceedings of the 7th ACM Conference on Electronic Commerce*, EC '06, pages 82–90, New York, NY, USA, 2006. ACM.

[76] James Pita, Manish Jain, Janusz Marecki, Fernando Ordóñez, Christopher Portway, Milind Tambe, Craig Western, Praveen Paruchuri, and Sarit Kraus. Deployed ARMOR protection: the application of a game theoretic model for security at the los angeles international airport. In *AAMAS 2008, Industry and Applications Track Proceedings*, pages 125–132, 2008.

[77] Danan Suryo Wicaksono and IA Karimi. Piecewise milp under-and overestimators for global optimization of bilinear programs. volume 54, pages 991–1008. Wiley Online Library, 2008.

[78] Kevin M. Carter, James F. Riordan, and Hamed Okhravi. A game theoretic approach to strategy determination for dynamic platform defenses. In *ACM MTD Workshop, 2014*, MTD '14. ACM, 2014.

[79] PratyusaK. Manadhata. Game theoretic approaches to attack surface shifting. In *Moving Target Defense II*, volume 100 of *AIS*, pages 1–13. Springer New York, 2013.

[80] Stephen Jones, Alexander Outkin, Jared Gearhart, Jacob Hobbs, John Siirola, Cindy Phillips, Stephen Verzi, Daniel Tauritz, Samuel Mulder, and Asmeret Naugle. Evaluating moving target defense with pladd. Technical report, Sandia National Labs-NM, 2015.

[81] Siv Hilde Houmb, Virginia NL Franqueira, and Erlend A Engum. Quantifying security risk level from cvss estimates of frequency and impact. *JSS*, 83(9):1622–1634, 2010.

[82] Craig H Rowland. Intrusion detection system, June 2002. US Patent 6,405,318.

[83] Douglas J Brown, Bill Suckow, and Tianqiu Wang. A survey of intrusion detection systems. *Department of Computer Science, University of California, San Diego*, 2002.

[84] Tal Garfinkel, Mendel Rosenblum, et al. A virtual machine introspection based architecture for intrusion detection. In *Ndss*, volume 3, pages 191–206, 2003.

[85] Amir Vahid Dastjerdi, Kamalrulnizam Abu Bakar, and Sayed Gholam Hassan Tabatabaei. Distributed intrusion detection in clouds using mobile agents. In *Advanced Engineering Computing and Applications in Sciences, 2009. ADVCOMP'09. Third International Conference on*, pages 175–180. IEEE, 2009.

[86] Laheeb Mohammad Ibrahim. Anomaly network intrusion detection system based on distributed time-delay neural network (dtdnn). *Journal of Engineering Science and Technology*, 5(4):457–471, 2010.

[87] Omar Al-Jarrah and Ahmad Arafat. Network intrusion detection system using neural network classification of attack behavior. *Journal of Advances in Information Technology Vol*, 6(1), 2015.

[88] Seongrae Jo, Haengnam Sung, and Byunghyuk Ahn. A comparative study on the performance of intrusion detection using decision tree and artificial neural network models. *Journal of the Korea Society of Digital Industry and Information Management*, 11(4):33–45, 2015.

[89] Sebastian Roschke, Feng Cheng, and Christoph Meinel. An extensible and virtualization-compatible ids management architecture. In *Information Assurance and Security, 2009. IAS'09. Fifth International Conference on*, volume 2, pages 130–134. IEEE, 2009.

[90] Aman Bakshi and Yogesh B Dujodwala. Securing cloud from ddos attacks using intrusion detection system in virtual machine. In *Communication Software and Networks, 2010. ICCSN'10. Second International Conference on*, pages 260–264. IEEE, 2010.

[91] Kleber Vieira, Alexandre Schulter, Carlos Westphall, and Carla Westphall. Intrusion detection for grid and cloud computing. *It Professional*, 12(4):38–43, 2010.

[92] Prachi Deshpande, SC Sharma, SK Peddoju, and S Junaid. Hids: A host based intrusion detection system for cloud computing environment. *International Journal of System Assurance Engineering and Management*, pages 1–10, 2014.

[93] Hervé Debar, Marc Dacier, and Andreas Wespi. Towards a taxonomy of intrusion-detection systems. *Computer Networks*, 31(8):805–822, 1999.

[94] Somesh Jha, Oleg Sheyner, and Jeannette Wing. Two formal analyses of attack graphs. In *Computer Security Foundations Workshop, 2002. Proceedings. 15th IEEE*, pages 49–63. IEEE, 2002.

[95] Deyan Chen and Hong Zhao. Data security and privacy protection issues in cloud computing. In *Computer Science and Electronics Engineering (ICCSEE), 2012 International Conference on*, volume 1, pages 647–651. IEEE, 2012.

[96] Colin Tankard. Advanced persistent threats and how to monitor and deter them. *Network security*, 2011(8):16–19, 2011.

[97] Martin Roesch et al. Snort: Lightweight intrusion detection for networks. In *Lisa*, volume 99, pages 229–238, 1999.

[98] Nizar Kheir, Nora Cuppens-Boulahia, Frédéric Cuppens, and Hervé Debar. A service dependency model for cost-sensitive intrusion response. In *European Symposium on Research in Computer Security*, pages 626–642. Springer, 2010.

[99] Mandiant Intelligence Center. Apt1: Exposing one of chinas cyber espionage units. *Mandian. com*, 2013.

[100] Peter Mell, Karen Scarfone, and Sasha Romanosky. Common vulnerability scoring system. *IEEE Security & Privacy*, 4(6), 2006.

[101] Christopher Kiekintveld, Manish Jain, Jason Tsai, James Pita, Fernando Ordóñez, and Milind Tambe. Computing optimal randomized resource allocations for massive security games. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 689–696. International Foundation for Autonomous Agents and Multiagent Systems, 2009.

[102] Dmytro Korzhyk, Vincent Conitzer, and Ronald Parr. Complexity of computing optimal stackelberg strategies in security resource allocation games. In *AAAI*, 2010.

[103] Praveen Paruchuri, Jonathan P Pearce, Janusz Marecki, Milind Tambe, Fernando Ordonez, and Sarit Kraus. Playing games for security: An efficient exact algorithm for solving bayesian stackelberg games. In *AAMAS*, 2008.

[104] Eric Budish, Yeon-Koo Che, Fuhito Kojima, and Paul Milgrom. Designing random allocation mechanisms: Theory and applications. *American Economic Review*, 103(2):585–623, 2013.

[105] Koral Ilgun and A USTAT. A real-time intrusion detection system for unix. *University of California Santa Barbara Master Thesis*, 1992.

[106] National vulnerability database. https://nvd.nist.gov. Accessed: 2018-09-25.

[107] Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Eleventh International Conference on Machine Learning*, 1994.

[108] Dmytro Korzhyk, Zhengyu Yin, Christopher Kiekintveld, Vincent Conitzer, and Milind Tambe. Stackelberg vs. Nash in Security Games: An Extended Investigation of Interchangeability, Equivalence, and Uniqueness. *J. Artif. Int. Res.*, 41(2):297–327, May 2011.

[109] Chun-Jen Chung, Pankaj Khatkar, Tianyi Xing, Jeongkeun Lee, and Dijiang Huang. Nice: Network intrusion detection and countermeasure selection in virtual network systems. *IEEE transactions on dependable and secure computing*, 10(4):198–211, 2013.

[110] Lloyd S Shapley. Stochastic games. *Proceedings of the national academy of sciences*, 39(10):1095–1100, 1953.

[111] Anjon Basak, Jakub Černỳ, Marcus Gutierrez, Shelby Curtis, Charles Kamhoua, Daniel Jones, Branislav Bošanskỳ, and Christopher Kiekintveld. An initial study of targeted personality models in the flipit game. In *International Conference on Decision and Game Theory for Security*, pages 623–636. Springer, 2018.

[112] Daniel Guerrero, Alin A Carsteanu, Rocio Huerta, and Julio B Clempner. An iterative method for solving stackelberg security games: A markov games approach. In *Electrical Engineering, Computing Science and Automatic Control (CCE), 2017 14th International Conference on*, pages 1–6. IEEE, 2017.

[113] Nagios Enterprises. Nagios, 2015.

[114] Dmytro Korzhyk, Zhengyu Yin, Christopher Kiekintveld, Vincent Conitzer, and Milind Tambe. Stackelberg vs. nash in security games: An extended investigation of interchangeability, equivalence, and uniqueness. *Journal of Artificial Intelligence Research*, 41:297–327, 2011.

[115] Western Region Cybersecurity Defense Competition. WRCCDC. https://archive.wrccdc.org/images/2018/, 2018.

[116] Ankur Chowdhary, Vaibhav Hemant Dixit, Naveen Tiwari, Sukhwa Kyung, Dijiang Huang, and Gail-Joon Ahn. Science dmz: Sdn based secured cloud testbed. In *Network Function Virtualization and Software Defined Networks (NFV-SDN), 2017 IEEE Conference on*, pages 1–2. IEEE, 2017.

[117] Zhisheng Hu, Minghui Zhu, and Peng Liu. Online algorithms for adaptive cyber defense on bayesian attack graphs. 2017.

[118] Arunesh Sinha, Thanh H Nguyen, Debarun Kar, Matthew Brown, Milind Tambe, and Albert Xin Jiang. From physical security to cybersecurity. *Journal of Cybersecurity*, 1(1):19–35, 2015.

[119] James Pita, Manish Jain, Janusz Marecki, Fernando Ordóñez, Christopher Portway, Milind Tambe, Craig Western, Praveen Paruchuri, and Sarit Kraus.

Deployed armor protection: the application of a game theoretic model for security at the los angeles international airport. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: industrial track*, pages 125–132. AAMAS, 2008.

[120] Manish Jain, Erim Kardes, Christopher Kiekintveld, Fernando Ordónez, and Milind Tambe. Security games with arbitrary schedules: A branch and price approach. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[121] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, et al. B4: Experience with a globally-deployed software defined wan. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 3–14. ACM, 2013.

[122] Joshua Letchford and Yevgeniy Vorobeychik. Optimal interdiction of attack plans. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 199–206. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

[123] Karel Durkota, Viliam Lisỳ, Branislav Bosanskỳ, and Christopher Kiekintveld. Optimal network security hardening using attack graph games. In *IJCAI*, pages 526–532, 2015.

[124] Swetasudha Panda and Yevgeniy Vorobeychik. Near-optimal interdiction of factored mdps. In *Conference on Uncertainty in Artificial Intelligence*, 2017.

[125] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[126] Joshua Letchford and Vincent Conitzer. Solving security games on graphs via marginal probabilities. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

[127] Ville Könönen. Asymmetric multiagent reinforcement learning. *Web Intelligence and Agent Systems: An international journal*, 2(2):105–121, 2004.

[128] National vulnerability database. https://nvd.nist.gov. Accessed: 2020-4-23.

[129] Csaba Szepesvári and Michael L Littman. A unified analysis of value-function-based reinforcement-learning algorithms. *Neural computation*, 11(8):2017–2060, 1999.

[130] Joel Brynielsson and Stefan Arnborg. Bayesian games for threat prediction and situation analysis. In *International Conference on Information Fusion*. Citeseer, 2004.

[131] Junling Hu, Michael P Wellman, et al. Multiagent reinforcement learning: theoretical framework and an algorithm. In *ICML*, volume 98, pages 242–250. Citeseer, 1998.

[132] Yoav Shoham, Rob Powers, and Trond Grenager. Multi-agent reinforcement learning: a critical survey. *Web manuscript*, 2003.

[133] Michael L Littman. Friend-or-foe q-learning in general-sum games. In *ICML*, volume 1, pages 322–328, 2001.

[134] Amy Greenwald, Keith Hall, and Roberto Serrano. Correlated q-learning. In *ICML*, volume 20, page 242, 2003.

[135] Willemien Kets. Finite depth of reasoning and equilibrium play in games with incomplete information. Technical report, Discussion Paper, Center for Mathematical Studies in Economics and ..., 2013.

[136] Muthukumaran Chandrasekaran, Yingke Chen, and Prashant Doshi. On markov games played by bayesian and boundedly-rational players. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[137] Harold William Kuhn and Albert William Tucker. *Contributions to the Theory of Games*, volume 2. Princeton University Press, 1953.

[138] Liam MacDermed, Charles Isbell, and Lora Weiss. Markov games of incomplete information for multi-agent reinforcement learning. In *Workshops at the Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

[139] Kurian Tharakunnel and Siddhartha Bhattacharyya. Leader-follower semi-markov decision problems: theoretical framework and approximate solution. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 111–118. IEEE, 2007.

[140] Chi Cheng, Zhangqing Zhu, Bo Xin, and Chunlin Chen. A multi-agent reinforcement learning algorithm based on stackelberg game. In *2017 6th Data Driven Control and Learning Systems (DDCLS)*, pages 727–732. IEEE, 2017.

[141] Régis Sabbadin and Anne-France Viet. Leader-follower mdp models with factored state space and many followers-followers abstraction, structured dynamics and state aggregation. In *Proceedings of the Twenty-second European Conference on Artificial Intelligence*, pages 116–124. IOS Press, 2016.

[142] Tianmin Shu and Yuandong Tian. M^3rl: Mind-aware multi-agent management reinforcement learning. *arXiv preprint arXiv:1810.00147*, 2018.

[143] Zhenyu Shi, Runsheng Yu, Xinrun Wang, Rundong Wang, Youzhi Zhang, Hanjiang Lai, and Bo An. Learning expensive coordination: An event-based deep rl approach. In *International Conference on Learning Representations*, 2019.

[144] Frederico Araujo, Kevin W. Hamlen, Sebastian Biedermann, and Stefan Katzenbeisser. From patches to honey-patches: Lightweight attacker misdirection, deception, and disinformation. In *21st ACM conf on Computer and Communications Security (CCS)*, pages 942–953, 2014.

[145] Marius Musch, Martin Härterich, and Martin Johns. Towards an automatic generation of low-interaction web application honeypots. In *Proc of the 13th International Conf on Availability, Reliability and Security (ARES)*, 2008.

[146] Corrado Leita, Ken Mermoud, and Marc C. Dacier. ScriptGen: An automated script generation tool for Honeyd. In *Proc of the 21st Conference on Annual Computer Security Applications(ACSAC)*, pages 203–214, 2005.

[147] Daniel Fraunholz, Daniel Reti, Simon Duque Anton, and Hans Dieter Schotten. Cloxy: A context-aware deception-as-a-service reverse proxy for web services. In *Proc of the 5th ACM Workshop on Moving Target Defense (MTD)*, pages 40–47, 2018.

[148] William A. Arbaugh, William L. Fithen, and John McHugh. Windows of vulnerability: A case study analysis. *IEEE Computer*, 33(12):52–59, 2000.

[149] Richard Lippmann, Seth Webster, and Douglas Stetson. The effect of identifying vulnerabilities and patching software on the utility of network intrusion detection. In *Proc of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID)*, pages 307–326, 2002.

[150] Eric Rescorla. Is finding security holes a good idea? *IEEE Security & Privacy*, 3(1):14–19, 2005.

[151] Hilary K. Browne, William A. Arbaugh, John McHugh, and William L. Fithen. A trend analysis of exploitations. In *Proc of the 22nd IEEE Symposium on Security & Privacy (S&P)*, pages 214–229, 2001.

[152] NIST. Vulnerability summary for CVE-2014-6277, 2014.

[153] David Roundy. Darcs: Distributed version management in Haskell. In *Proc of the ACM SIGPLAN Workshop on Haskell*, pages 1–4, 2005.

[154] Frederico Araujo, Mohammad Shapouri, Sonakshi Pandey, and Kevin Hamlen. Experiences with honey-patching in active cyber security education. In *proc of the 8th work Cyber Security Experimentation and Test (CSET*, 2015.

[155] Frederico Araujo and Kevin W. Hamlen. Compiler-instrumented, dynamic secret-redaction of legacy processes for attacker deception. In *24th USENIX Security Symp.*, pages 145–159, 2015.

[156] Ian Lynagh. An algebra of patches. Technical report, University of Oxford, 2006.

[157] Darcs. Darcs version control system, 2016.

[158] Toolswatch. vFeed – The correlated vulnerability and threat database, 2016.

[159] NIST. CVSS. https://www.first.org/cvss, 2016. [19-Nov-2016].

[160] Py-idstools. Py-idstools Python library, 2016.

[161] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[162] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *CVPR*, 2016.

[163] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *EuroS&P*, pages 372–387. IEEE, 2016.

[164] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *ACM CCS*, 2017.

[165] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. *arXiv:1708.03999*, 2017.

[166] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. *arXiv:1705.07204*, 2017.

[167] Stephan Zheng, Yang Song, Thomas Leung, and Ian Goodfellow. Improving the robustness of deep neural networks via stability training. In *CVPR*, 2016.

[168] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael Wellman. Towards the science of security and privacy in machine learning. *arXiv:1611.03814*, 2016.

[169] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *IEEE S&P*, 2016.

[170] Arjun Nitin Bhagoji, Daniel Cullina, and Prateek Mittal. Dimensionality reduction as a defense against evasion attacks on machine learning classifiers. *CoRR*, abs/1704.02654, 2017.

[171] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *IEEE S&P*, 2017.

[172] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. *arXiv:1610.08401*, 2016.

[173] Tom B Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. Adversarial patch. *arXiv:1712.09665*, 2017.

[174] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv:1708.06733*, 2017.

[175] Battista Biggio, Giorgio Fumera, and Fabio Roli. Adversarial pattern classification using multiple classifiers and randomisation. *Structural, Syntactic, and Statistical Pattern Recognition*, 2008.

[176] Yevgeniy Vorobeychik and Bo Li. Optimal randomized classification in adversarial settings. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 485–492. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

[177] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv:1502.03167*, 2015.

[178] Mahdieh Abbasi and Christian Gagné. Robustness to adversarial examples through an ensemble of specialists. *arXiv:1702.06856*, 2017.

[179] Warren He, James Wei, Xinyun Chen, Nicholas Carlini, and Dawn Song. Adversarial example defenses: Ensembles of weak defenses are not strong. *arXiv preprint arXiv:1706.04701*, 2017.

[180] George A Adam, Petr Smirnov, Anna Goldenberg, David Duvenaud, and Benjamin Haibe-Kains. Stochastic combinatorial ensembles for defending against adversarial examples. *arXiv:1808.06645*, 2018.

[181] Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya Nori, and Antonio Criminisi. Measuring neural net robustness with constraints. In *NIPS*, 2016.

[182] Tsui-Wei Weng, Huan Zhang, Pin-Yu Chen, Jinfeng Yi, Dong Su, Yupeng Gao, Cho-Jui Hsieh, and Luca Daniel. Evaluating the robustness of neural networks: An extreme value theory approach. *arXiv preprint arXiv:1801.10578*, 2018.

[183] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[184] Yong Du, Wei Wang, and Liang Wang. Hierarchical recurrent neural network for skeleton based action recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1110–1118, 2015.

[185] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the devil in the details: Delving deep into convolutional nets. *arXiv:1405.3531*, 2014.

[186] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *ACM international conference on Multimedia*.

[187] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[188] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556*, 2014.

[189] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[190] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[191] Southwest blackout. https://tinyurl.com/y6xxjsm5. accessed: 2020-06-30.

[192] Vahid Salehi, Ahmed Mohamed, Ali Mazloomzadeh, and Osama A Mohammed. Laboratory-based smart power system, part ii: Control, monitoring, and protection. *IEEE Transactions on Smart Grid*, 3(3):1405–1417, 2012.

[193] Anamitra Pal, Anil Kumar S Vullikanti, and Sekharipuram S Ravi. A pmu placement scheme considering realistic costs and modern trends in relaying. *IEEE Transactions on Power Systems*, 32(1):552–561, 2016.

[194] Kaustav Basu, Malhar Padhee, Sohini Roy, Anamitra Pal, Arunabha Sen, Matthew Rhodes, and Brian Keel. Health monitoring of critical power system equipments using identifying codes. In *CRITIS*, pages 29–41. Springer, 2018.

[195] Malhar Padhee, Reetam Sen Biswas, Anamitra Pal, Kaustav Basu, and Arunabha Sen. Identifying unique power system signatures for determining vulnerability of critical power system assets. *ACM SIGMETRICS Performance Evaluation Review*, 47(4):8–11, 2020.

[196] Stamatis Karnouskos. Stuxnet worm impact on industrial cyber-physical system security. In *Annual Conference of the IEEE Industrial Electronics Society*, 2011.

[197] Symantec Team. Dragonfly: Western energy sector targeted by sophisticated attack group, 2017.

[198] Sai Pushpak Nandanoori, Soumya Kundu, Seemita Pal, Khushbu Agarwal, and Sutanay Choudhury. Model-agnostic algorithm for real-time attack identification in power grid using koopman modes. *Arxiv 2007.11717*, 2020.

[199] Luyao Niu and Andrew Clark. A framework for joint attack detection and control under false data injection. In *International Conference on Decision and Game Theory for Security*, pages 352–363. Springer, 2019.

[200] Mark G Karpovsky, Krishnendu Chakrabarty, and Lev B Levitin. On a new class of codes for identifying vertices in graphs. *IEEE Transactions on Information Theory*, 1998.

[201] Emmanuel Charbit, Irene Charon, Gérard Cohen, and Olivier Hudry. Discriminating codes in bipartite graphs. *Electronic Notes in Discrete Mathematics*, 26:29–35, 2006.

[202] Ray Daniel Zimmerman, Carlos Edmundo Murillo-Sánchez, and Robert John Thomas. Matpower: Steady-state operations, planning, and analysis tools for power systems research and education. *IEEE Transactions on power systems*, 26(1):12–19, 2010.

[203] Deepjyoti Deka, Ross Baldick, and Sriram Vishwanath. Optimal data attacks on power grids: Leveraging detection & measurement jamming. In *2015 IEEE International Conference on Smart Grid Communications*. IEEE, 2015.

[204] Ruilong Deng, Gaoxi Xiao, Rongxing Lu, Hao Liang, and Athanasios V Vasilakos. False data injection on state estimation in power systems—attacks, impacts, and defense: A survey. *IEEE Transactions on Industrial Informatics*, 13(2):411–423, 2016.

[205] Song Tan, Debraj De, Wen-Zhan Song, Junjie Yang, and Sajal K Das. Survey of security advances in smart grid: A data driven approach. *IEEE Communications S&T*, 2017.

[206] Brycent Chatfield and Rami J Haddad. Moving target defense intrusion detection system for ipv6 based smart grid advanced metering infrastructure. In *SoutheastCon 2017*.

[207] Bradley Potteiger, Feiyang Cai, Abhishek Dubey, Xenofon Koutsoukos, and Zhenkai Zhang. Security in mixed time and event triggered cyber-physical systems using moving target defense. In *IEEE International Symposium on Real-Time Distributed Computing*. IEEE, 2020.

[208] Kaustav Basu, Sanjana Dey, Subhas Nandy, and Arunabha Sen. Sensor networks for structural health monitoring of critical infrastructures using identifying codes. In *DRCN*. IEEE, 2019.

[209] Sailik Sengupta, Ankur Chowdhary, Abdulhakim Sabur, Adel Alshamrani, Dijiang Huang, and Subbarao Kambhampati. A survey of moving target defenses for network security. *IEEE Communications Surveys & Tutorials*, 2020.

[210] Emmanuel Hebrard, Brahim Hnich, Barry O'Sullivan, and Toby Walsh. Finding diverse and similar solutions in constraint programming. In *AAAI*, volume 5, pages 372–377, 2005.

[211] Zahra Zahedi, Sailik Sengupta, and Subbarao Kambhampati. Why not give this work to them?'explaining ai-moderated task-allocation outcomes using negotiation trees. *arXiv preprint arXiv:2002.01640*, 2020.

[212] Sailik Sengupta, Tathagata Chakraborti, Sarath Sreedharan, Satya Gautam Vadlamudi, and Subbarao Kambhampati. Radar—a proactive decision support system for human-in-the-loop planning. In *2017 AAAI Fall Symposium Series*, 2017.

[213] Sailik Sengupta, Tathagata Chakraborti, and Subbarao Kambhampati. Maradar–a mixed-reality interface for collaborative decision making. *ICAPS UISP*, 2018.

[214] Sachin Grover, Sailik Sengupta, Tathagata Chakraborti, Aditya Prasad Mishra, and Subbarao Kambhampati. ipass: A case study of the effectiveness of automated planning for decision support. 2019.

[215] Sachin Grover, Sailik Sengupta, Tathagata Chakraborti, and Subbarao Kambhampati. Radar—a proactive decision support system for human-in-the-loop planning. 2020.

[216] Aditya Prasad Mishra, Sailik Sengupta, Sarath Sreedharan, Tathagata Chakraborti, and Subbarao Kambhampati. Cap: A decision support system for crew scheduling using automated planning.

[217] Valmeekam Karthik, Sarath Sreedharan, Sailik Sengupta, and Subbarao Kambhampati. Radar-x: An interactive interface pairing contrastive explanations with revised plan suggestions. In *Technical Report (Under Review)*, 2020.

[218] Alberto Olmo, Sailik Sengupta, and Subbarao Kambhampati. Not all failure modes are created equal: Training deep neural networks for explicable (mis) classification. *arXiv preprint arXiv:2006.14841*, 2020.

[219] Tathagata Chakraborti, Sarath Sreedharan, Sailik Sengupta, TK Satish Kumar, and Subbarao Kambhampati. Compliant conditions for polynomial time approximation of operator counts. In *Ninth Annual Symposium on Combinatorial Search*, 2016.

[220] Sailik Sengupta, He He, Batool Haider, Spandana Gella, and Mona Diab. Natural language generation with keyword constraints– a hybrid approach using supervised and reinforcement learning. *West-Coast Conference on Natural Language Processing*, 2019.

[221] Niharika Jain, Alberto Olmo, Sailik Sengupta, Lydia Manikonda, and Subbarao Kambhampati. Imperfect imaganation: Implications of gans exacerbating biases on facial data augmentation and snapchat selfie lenses. *arXiv preprint arXiv:2001.09528*, 2020.

[222] Thomas B Sheridan and William L Verplank. Human and computer control of undersea teleoperators. Technical report, 1978.

[223] Raja Parasuraman. Designing automation for human use: empirical studies and quantitative models. *Ergonomics*, 2000.

[224] Dongyu Meng and Hao Chen. Magnet: a two-pronged defense against adversarial examples. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 135–147, 2017.

[225] Nicholas Carlini and David Wagner. Magnet and" efficient defenses against adversarial attacks" are not robust to adversarial examples. *arXiv preprint arXiv:1711.08478*, 2017.

[226] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*, 2018.

[227] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

[228] Kuan-Chieh Wang, Paul Vicol, James Lucas, Li Gu, Roger Grosse, and Richard Zemel. Adversarial distillation of bayesian neural network posteriors. *arXiv preprint arXiv:1806.10317*, 2018.

[229] Neale Ratzlaff and Li Fuxin. Hypergan: A generative model for diverse, performant neural networks. *arXiv preprint arXiv:1901.11058*, 2019.

[230] Shixin Tian, Guolei Yang, and Ying Cai. Detecting adversarial examples through image transformation. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[231] Eitan Farchi, Onn Shehory, and Guy Barash. Defending via strategic ml selection. *arXiv preprint arXiv:1904.00737*, 2019.

[232] Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. Synthesizing robust adversarial examples. *arXiv preprint arXiv:1707.07397*, 2017.

[233] Chawin Sitawarin, Arjun Nitin Bhagoji, Arsalan Mosenia, Prateek Mittal, and Mung Chiang. Rogue signs: Deceiving traffic sign recognition with malicious ads and logos. *arXiv preprint arXiv:1801.02780*, 2018.

[234] Gamaleldin Elsayed, Shreya Shankar, Brian Cheung, Nicolas Papernot, Alexey Kurakin, Ian Goodfellow, and Jascha Sohl-Dickstein. Adversarial examples that fool both computer vision and time-limited humans. In *Advances in Neural Information Processing Systems*, pages 3910–3920, 2018.

APPENDIX A

OTHER RESEARCH CONTRIBUTIONS

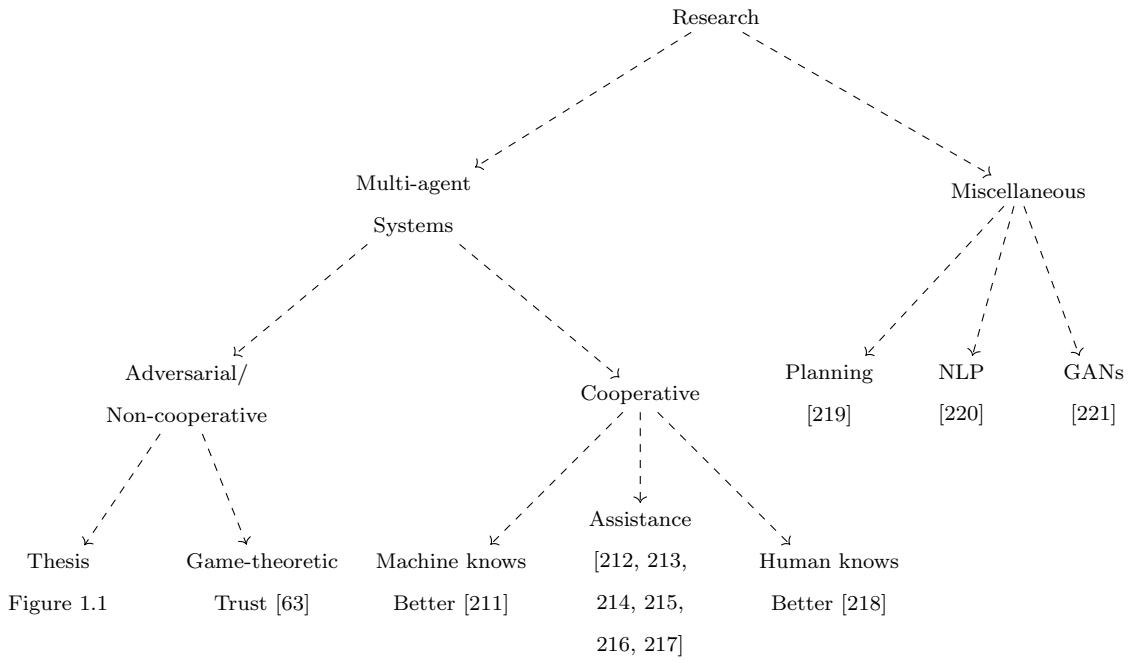


Figure A.1: An overview of the my research contributions.

The thesis comprises of works that model Moving Target Defense as a multi-agent interaction. By virtue of it being a *defense* mechanism, the problems we looked at had an adversarial element. Beyond these scenarios, I have been fortunate enough to collaborate on a variety of other problems in multi-agent systems and other topics. An overview of my research as a graduate student alongside is given in Figure A.1. I situate the work discussed in my thesis under the multi-agent adversarial/non-cooperative scenario. In this appendix, I will briefly given an overview of the other contributions.

Game-Theoretic Trust The work closest to the scenarios described in the thesis is the formulation of human-robot interaction as a game where the human acts as a supervisor and seeks to come up with a probabilistic supervision mechanism that (1) incentivizes a worker robot to follow all protocols and (2) reduces supervision costs [63].

Multi-agent Co-operative Settings

We divide the works using the knowledge asymmetry between an AI-agent and a human-in-the-loop. In the spectrum of machine knows more (than the human) and human knows more (than the machine), we situate our work on developing decision support assistance.

Machine Knows More Task-allocation is an important problem in multi-agent systems. It becomes more challenging when the team-members are humans with imperfect knowledge about their teammates' costs and the overall performance metric. While distributed task-allocation methods lets the team-members engage in iterative dialog to reach a consensus, the process can take a considerable amount of time and communication. On the other hand, a centralized method that simply outputs an allocation may result in discontented human team-members who, due to their imperfect knowledge and limited computation capabilities, perceive the allocation to be unfair. To address these challenges, in [211], we proposed a centralized Artificial Intelligence Task Allocation (AITA) that simulated a negotiation and produced a negotiation-aware task allocation that was fair. If a disgruntled team-member was unhappy with the proposed allocation, we allowed them to question the proposed allocation using a counterfactual. By using parts of the simulated negotiation, we were able to provide contrastive explanations that providing minimum information about other's cost to refute their foil.

Human Knows More Deep Neural Networks are often brittle on image classification tasks and known to misclassify inputs. While these misclassifications may be inevitable, all failure modes cannot be considered equal. Certain misclassifications

(eg. classifying the image of a dog to an airplane) can perplex humans and result in the loss of human trust in the system. Even worse, these errors (eg. a person misclassified as a primate) can have odious societal impacts. Thus, in in [218], we tried to reduce inexplicable errors. First, we proposed methods to obtain the class-level semantics that captures the human's expectation (M^h) regarding which classes are semantically close *vs.* ones that are far away. Second, we proposed the use of Weighted Loss Functions (WLFs) to penalize misclassifications by the weight of their inexplicability. Finally, we showed that training (or even fine-tuning) existing classifiers with the two proposed methods lead to Deep Neural Networks that have (1) comparable top-1 accuracy, an important metric in operational contexts, (2) more explicable failure modes, (3) higher robustness to random and adversarial noise and (4) incur significantly less cost in gathering of additional human labels compared to existing works.

Assistance Proactive Decision Support (PDS) aims at improving the decision making experience of *human decision makers* by enhancing both the quality of the decisions and the ease of making them. In [20], we considered the role of automated decision making technologies in the deliberative process of the human decision maker. Specifically, we focused on expert humans in the loop who shared a detailed, if not complete, model of the domain with the assistant, but may still be unable to compute plans due to cognitive overload. To this end, we proposed a PDS framework `RADAR`. By leveraging technologies in the automated planning community, `RADAR` aided the human decision maker in constructing plans, validating and correcting them, suggesting completions, and explaining its suggestions, etc. We engineered the capabilities of such a system based on good design principles laid out in the literature on human-computer interaction; we leveraged the levels of automation defined in [222], and the

stages of automation [223]. In [214], via human subject studies, we showed that the time taken to make plans drastically reduced and the satisfaction improves when humans use the capabilities of our decision support system. In this setting, we enabled RADAR to provide explanations when the human is not aware of certain domain rules. When such explanations seemed unreasonable and the human, the longitudinal interaction was be leveraged to correct RADAR's own model of the world or elicit human's hidden preferences [217]. Thus, the work on assistance falls between the two scenarios described above. Given the human is to be held responsible for the final plan, an asymmetry exists, making this more of decision support as opposed to human-AI teaming. We adapt our notion of decision support to support teams that have individuals with varied capabilities [213] and task-scheduling problems [216].

Miscellaneous

These works can be divided into three branches. In planning, I worked on developing a computationally simpler version of the operator-count heuristic for a particular class of planning domains [219]. In Natural Language Processing, I formalized the problem of sentence generation from keywords and built end-to-end neural models that, trained using supervised and reinforcement learning methods, were able to outperform sampling based methods [220]. Finally, I tried to understanding the propagation of biases in synthetic data generation using Generative Adversarial Networks (GANs). In [221], we showed that mode-collapse inhibits the capability of GANs to not merely propagate biases in existing data, but exacerbate them. Our recent results show that addressing the mode-collapse issue using state-of-the-art mechanisms fail to address the challenge along certain embargoed features. A similar phenomenon of exacerbating societal biases is also observed in the context of conditional GANs.

APPENDIX B

SIMULATION ENVIRONMENT DESCRIPTION

In this chapter, we first describe the Open-AI style game-simulator interface that can be used by the learning agents in Chapter 5. We then briefly describe how some of the simulator functionalities are provided for the domains and discuss future directions.

```
def get_states():
 ...
 return []
"""
@Input
 None
@Output
 Returns a list (essentially a set) consisting of states in the game
    .
"""

def get_start_state():
 ...
 return s
"""
@Input
 None
@Output
 A start state s ∈ start_S ⊆ S denoting the start state for an episode.
"""

def get_actions():
 ...
 (return [[ [], [], ... ], []])
"""
@Input
```

```
 None
@Output
 A list with the first element representing attacker actions and the
    second element representing defender actions. The first list can
    be further decomposed in to set of lists representing actions
   for each attacker/follower type.
"""

def is_end(s):
 ...
 return True/False
"""
@Input
 A state s ∈ S.
@Output
 AssertionError if s ∉ S.
 True if the input state s ∈ end_S ⊆ S
 False otherwise
"""

def act(s, a_D, a_A, θ):
 ...
 return R_D, R_A, s_{t+1}
"""
@Input
 A state s ∈ S, defender's action a_D, attacker/follower's action a_A,
   the attacker type θ
@Output
 AssertionError if s ∉ S, a_D (or a_A) is not a defender (or attacker
   type θ's) action.
 R_D -- Defender's utility
```

```
52 R_A -- Attacker's utility
53 s_{t+1} -- Next state
54 """
```

Moving Target Defense for Web-applications

As mentioned earlier, we borrow the game domain from [20] to build the simulator. In the context of the simulator functions, the `get_state` method returns four states of the system. Each state of the defender constitutes choosing an implementation language (Php or python) and a database technology (SQL or PostgreSQL) that can be used to host the web-application.

The `get_start_state` method of the game simulator returns on of the four states at random, implying the system can start in any one of the four configurations. We allow a user to override the global variable that describes the set of start states because, in the context of certain baseline strategies such as BSG formulation [20], they consider only a sub-set of the MTD configurations. Thus, a strategy that places zero probability of switching to a configuration can new start or find itself in the configuration.

The `get_actions` method returns the set of actions available to the attacker types and the defender. Similar to the original game designed in [20], we pair up an attacker type expertise level with a set of technologies it has expertise in to the Common Vulnerabilities and Exploits (CVEs) mined from the NVD database to define their attack set. For the defender, the four configurations it can choose constitute the attack set.

The `is_end` always returns False in this setting as the game has not terminal state and the goal of the defender is to continuously keep moving the system. While we can stop when policy converges for our propose BSS-Q algorithm, for the other algorithm, owing to the lack of convergence guarantees, we run it for a predefined number of episodes.

The `act` method considers the action of the defender and the attacker's actions to determine the impact. Ideally, this can be done by deploying the system on a new configuration and then sending out an actual request to the web-service with the attack folder as part of a request. Then, depending on the return, decide if the attack was successful or not. While we seek to generate a class of attacks that can represent the 300+ CVEs used in the domain, this was out of scope for this work. Further, the attack success or failure might only give a binary indication of the rewards fro the attacker, which constitutes a sparse signal and treats impactful and trivial attacks under the same umbrella. To address this, we leverage the Common Vulnerability Scoring Service (CVSS) and use the Impact score as the attacker type reward if the chosen attack is expected to work on the defense configuration being deployed. We use the current state of the system and the defender action to compute the cost of the movement. Ideally, we want to determine this by running a system with multiple virtual machines– one hosting the current configuration and the other bringing up the next configuration (determined by $a_{\mathcal{D}}$). The time taken in bringing up the new configuration and then the amount of packets dropped or the extra resources used in the switching should all be part of the switching costs. While one can come up with an elaborate procedure to do so, this is beyond the present scope of our paper and we consider the costs calculated based on configuration-based similarity [20]– configurations that are more dissimilar incur higher switching costs.

Moving Target Defense for Placement of Intrusion Detection Systems in Cloud Networks

Before describing the design of the simulator, we provide the underlying cloud network, scenario, derived from the modeling in Section 4.2, in Figure B.1. This scenario, via two transformation steps– first, to an attack graphs and then to a Markov Game– can be used to build our game simulator. For details of this process, we refer the reader to look at Chapter 4.

The `get_start_state` method of the game simulator returns the single start state that represents the case where an attacker has user access to the LDAP server on the public facing interface of the network system.

The `get_actions` returns the set of attack actions, discovered using vulnerability scanners on the cloud system and a part of the attack graph representation in [42], that are possible for an attacker to execute in a given state of the cloud network system. The action set for the defender indicates the set of Intrusion Detection Systems they can place and a no-op action implying that they may not choose to place an IDS system at all.

The `is_end` returns a single state of the BSMG. This state represents the condition where the attacker has administrator access on the file server.

The `act` method considers the action of the defender and the attacker's actions to determine if the attack action is detected. Ideally, we want to play the defender's strategy of placing IDS system and then execute the attack action chosen by the attacker. While this needs an entire cloud network setup with Virtual Machines (VMs), we use similar resources used in [42] to determine the utilities and the transition. If the IDS placed is able to detect the exploit, we return a reward proportional to the

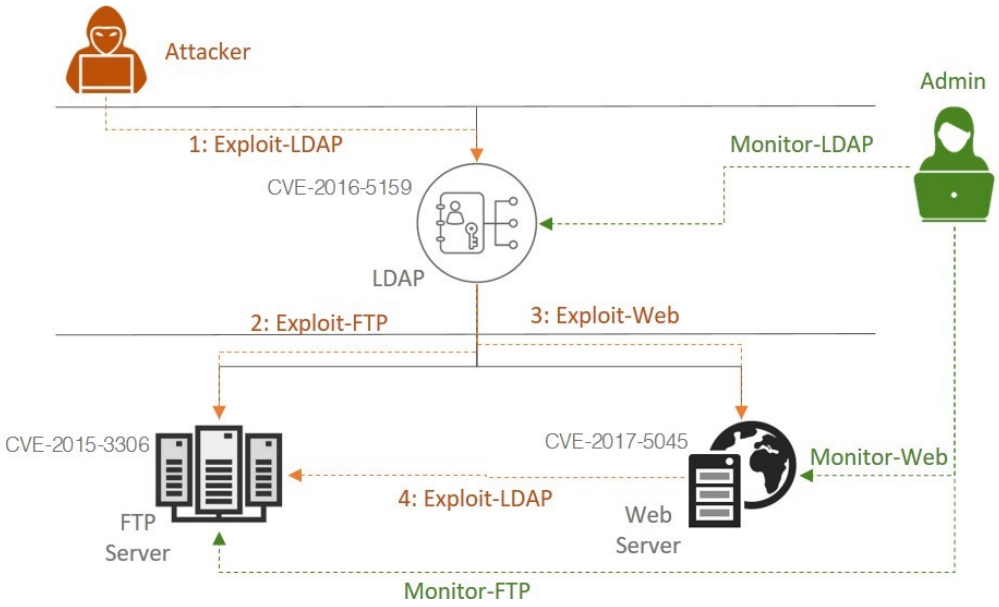


Figure B.1: A cloud system highlighting its network structure, the attacker and defender (admin) agents, the possible attacks and monitoring mechanisms.

effort required by the attacker in executing the attack action. This, similar to [42], is determined using the exploitability score of the CVE determined from the Common Vulnerability Scoring System (CVSS). If the IDS systems fails to detect the attack, then the attacker has an impact proportional to the base score obtained from CVSS which considers both the impact and the complexity of the attack vector. Further, the game transitions into a new state where the attacker has either escalated privileges on the same VM (or physical server) or gets access to a new VM on the cloud. The defender besides the impact score, which represents the security impact, considers the impact on network bandwidth if they deploy a Network Intrusion Detection System (NIDS) and impact on memory or CPU resources if they deploy a Host-based Intrusion Detection System (HIDS). We use the scaling used in [42] to come up with a single utility value for the defender.

APPENDIX C

OBTAINING DIFFERENTIAL IMMUNITY IN DEEP NEURAL NETWORKS

I will discuss three promising directions that can result in the development of differential immune networks– (1) carefully engineering (learnable and hyper) parameters of the neural networks, (2) modification of the input feature space and (3) analysis of the different attacks.

Engineering Neural Network Parameters

While our work in Chapter 7 tries to leverage the use of different architectures in in the hope of obtaining high differential immunity it is by no means a method that ensures it. In the hope of guaranteeing differential immunity, one can explore several directions to modify the parameters of neural networks; broadly, such modifications can be categorized into two types– (1) to tune the hyper-parameters of a network (eg. number of hidden layers, types of these layers, non-linear activation functions, etc.), and (2) the learnable parameters (weights and biases) of the network. In this section, we will discuss related work and promising avenues of research under these headings.

Engineering Hyper-parameters There exists work by [180] that considers the approach of modifying neural network architecture to design differentially immune networks. Their idea builds on top of MagNET [224], and our work MTDeep [17].

In MagNET, the authors use auto-encoders (AE) to both detect and reform adversarial inputs. The key assumption is that adversarial inputs are far away from benign test inputs in the data-manifold. Thus, reconstructions using AE, helps to discard malicious inputs with egregious perturbations (reconstruction error $>$ threshold) and refine those with sufficiently small perturbations (reconstruction error $<$ threshold but, malicious inputs are automatically mapped to the data manifold via reconstruction). While Carlini demonstrates that such a defense can be broken [225], the way it is used in [180] is slightly different. In [180], the authors consider placing

a Variational Auto-Encoder (VAE) before every layer of a network. Thus, detection and reform are performed on the transformed inputs of every layer.

If a network has $n-1$ convolutional layers followed by a fully connected layer, then there are n locations where a VAE can be added. Thus, a defender can switch between these 2^n networks at run-time (referred to as the pure-strategies/MTD configuration space in the MTDeep setting). They empirically demonstrate that differentially immunity exists between the various configurations and then provide justification as to why it exists. In LeNet, a 3-layer network architecture for classifying MNIST data, the different networks (called LeNet-VAE) can be denoted as $\{(0,0,0),(1,0,0),(0,1,0),\ldots,(1,1,1)\}$ where 1 indicates that a VAE is used before a particular layer. For example, the configuration $(1,0,0)$ indicates that VAE is only placed before the first convolutional layer to transform the input (but not before the second convolutional layer and the final fully-connected layer). They are able to show that attacks crafted for $(1,1,1)$, the strongest defense that detects and reforms examples before inputting them to any layer, do not transfer to single LeNet-VAE variants like $(0,1,0)$ or $(0,0,1)$.

It should seem counter-intuitive that attacks which can fool $(1,1,1)$ are not effective against $(0,1,0)$. The authors claim that, in the context of norm-based attacks that use gradient information, the gradients used to fool $(1,1,1)$ are orthogonal to the gradient required to fool $(0,1,0)$. To substantiate this claim, the authors compare the last layer's gradients of the various neural nets w.r.t. the input features. They show that the cosine distance similarity between gradients of the various networks is a strong predictor of the transferability of attacks. For example, the cosine distance similarity between the last layer gradients of $(1,1,1)$ and $(0,1,0)$ is small and thus, attack transferability is less, leading to greater differential immunity. To

strengthen this claim, the authors show that training an ensemble of eight networks to have orthogonal input-output Jacobians, compared to parallel input-output Jacobians, results in greater differential immunity (although such training may incur a loss in accuracy against benign test samples). Note that in [180], the authors first come up with the use of certain architectural modifications and discover that the different configurations have small cosine distance similarity between the last layer gradients, which in turn helps them develop differentially immune networks. Future works can use this insight to invent architectural modifications of neural networks to ensure differential immunity while limiting the accuracy loss against benign example.

On a slightly different note, there have been works on automatically discovering neural network architecture such as neural architecture search [226]. While these works have mostly been interested in guaranteeing high classification accuracy, changing their objectives to consider robustness to adversarially perturbed inputs can be a promising, albeit resource-intensive, direction. In the context of neural architecture search, actions and heuristics inspired from the findings of [180] (eg. different permutation of VAE placement before layers of a neural network can result in orthogonal gradients, and thus, higher differential immunity) can result in the discovery of an incumbent set of configurations with a predefined differential immunity (and minimum performance accuracy). Another interesting set of actions is the consideration of various non-linear activation functions for designing differentially immune configurations. An obvious downside to such a method is the time involved in obtaining this set.

Engineering weights and biases In this regard, works have considered methods to generate weights of a specified network [227, 228, 229]. Of special interest, is the work on HyperGANs [229]; the authors try to generate use-and-throw weights for a neural network using GANs while trying to ensure diversity, i.e. prevent mode-

collapse so that the GAN does not land up generating similar weights every time. To do so, they consider an intermediate noise vector that (1) has correlation across various dimensions that helps in generating parameters for a highly accurate neural network and (2) is similar to high entropy noise thereby preventing mode-collapse and ensuring diversity in the space of generated weights.

This method provides a computationally efficient way of generating tons of networks and picking a set of weights (or network configurations) that guarantee differential immunity. In this regard, the authors show that adversarial examples devised for one set of weights (i.e. with a white-box threat model) can only fool $50 - 70\%$ of the generated networks, but never all of them. Thus, this can greatly help in designing an ensemble with high differential immunity in the context of MTDeep.

Modification of the Input Feature Space In this case, one can consider randomly modifying the input features at test time (before inputting them to a network). Thus, the configuration of the MTDeep ensemble differs based on the input as opposed to the network (learnable or hyper) parameters.

In [230], the authors show that transformations such as rotation (clockwise or anti-clockwise) are extremely effective in negating the effect of adversarial noise while preserving the classification accuracy on benign examples. For example, a benign test image of a zero is classified to zero whether it is rotated or not whereas an adversarial example is correctly classified after transformation. Unfortunately, if an adversary is aware of the transformation technique in place, i.e. a white-box setting, strong attacks like the Carlini-Wagner attack designed in [171] can fool the network. On the other hand, when the authors consider a randomized rotation angle decided at test time before inputting the image to the network, they see an increase in robustness to adversarial attacks. This goes to say that white-box attacks designed for a predefined

rotation angle do not transfer to networks that use other rotation angles. Thus, investigation of various image transformation techniques can be a promising direction to generate differentially immune configurations.

Beyond popular transformation functions, one can consider partitioning techniques to carefully divide the input space. If the input feature space is denoted as F, one can consider the partition $\{F_1, F_2, \ldots, F_n\}$ such that $F_1 \cup F_2 \cup \cdots \cup F_n = F$ and $\forall\ i, j\ F_i \cap F_j = \emptyset$. For example, consider $n = 2$ and F_1 as the set of all pixels in the odd rows of an input image (and thus, F_2 the set of all pixels in the even rows). In such a setting, if we limit norm-based perturbations to be of a reasonable size, then two networks with F_1 and F_2 can produce differentially immune networks because white-box attacks on F_i cannot fool F_j when $i \neq j$ and the bound on the adversarial noise removes attacks that can perturb pixels in both the sets F_1 and F_2 from being considered a valid attack. The latter is not a strong assumption because detection methods can often be used to reliably detect and discard such examples. Some works build on top of MTDeep in this regard; in [231] the authors consider a simple problem with four input features and come up with a predefined partitioning strategy. By using a mixed Nash equilibrium strategy for randomization at test-time (an idea similar to MTDeep [17]), they are able to show the efficacy of their method against one-feature perturbation attacks.

Beyond simple partition methods, more intelligent techniques can be considered to balance the loss in accuracy of benign inputs while beginning effective against adversarial examples. For example, consider the use of using the gradient information of the network w.r.t. the input features (i.e. $\frac{\delta L}{\delta x}$ as opposed to $\frac{\delta L}{\delta \theta}$) to allocate features to the partitions sets. Each network, trained on the disjoint partitions, can then be strategically chosen at run-time, similar to MTDeep, for classifying an input image.

Analysing various attack types

Majority of the work in regards to the development of attacks and defenses in the context of Deep Neural Networks considers norm-based perturbations, i.e. strategic crafting and addition of noise to a benign image such that the malicious image is (1) within a norm-bound of the original image and (2) able to make the classifier misclassify it. The first restriction is often imposed to ensure that a human can correctly classify the image. Beyond these, there exists a realm of work on generating adversarial examples that are more practical; for example, robust 3D adversarial patches that when put on any object can fool the classifier regardless of noise due to camera angles, lighting conditions, etc. [232], the addition of stickers to traffic signs that seem inconsequential to a human but make the classifier behave incorrectly [233], adversarial examples that make a classifier and a (time-limited) human misclassify [234] etc.

Adversarial training, i.e. re-training the network with adversarial examples generated using a particular attack can often make the network immune to that attacks. I believe that adversarial training with all types of attack images (norm-based, adv. patches, etc.) will negatively impact classification accuracy (for example, a particular attack type, eg. attacks that increase the brightness of the image are in direct conflict with other attacks eg. attacks that darken the image or increase its contrast). Thus, it will be worth investigating if adversarially trained networks on one attack type are still prone to adversarial inputs crafted using other attack types. If they are, one case considers an ensemble of classifiers where each classifier is adversarially trained against one attack type. This will guarantee differential immunity because a defender has at least one configuration that is robust to every attack– the one adversarially trained on that attack.