2020-01-01

Abstraction Techniques In Security Games With Underlying Network Structure

Anjon Basak
University of Texas at El Paso

ABSTRACTION TECHNIQUES IN SECURITY GAMES WITH UNDERLYING NETWORK STRUCTURE

ANJON BASAK

Doctoral Program in Computer Science

APPROVED:

Christopher Kiekintveldt, Ph.D. Chair

Martine Ceberio, Ph.D.

Deepak Tosh, Ph.D.

Charles Kamhoua, Ph.D.

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

To My

Mother

With Love

ABSTRACTION TECHNIQUES IN SECURITY GAMES WITH UNDERLYING NETWORK STRUCTURE

by

ANJON BASAK

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

Doctoral Program in Computer Science

THE UNIVERSITY OF TEXAS AT EL PASO

May 2020

Abstract

In a multi-agent system, multiple intelligent agents interact with each other in an environment to achieve their objectives. They can do this because they know which actions are available to them and which actions they prefer to take in a particular situation. The job of game theory is to analyze the interactions of the intelligent agents by different solution techniques and provide analysis such as predicting outcomes or recommending courses of action to specific players. To do so game theory works with a model of real-world scenarios which helps us to make a better decision in our already complex daily life. Game theory also has a growing role in protecting us and our infrastructure, as well as protecting wildlife from crimes. In this thesis, I particularly focus on adversarial game theoretic models where the interacting agents' decision space involves network structure both in the physical domain and cyber domain. In this type of game model, the strategy space is typically represented by a network with interconnected nodes representing valuable entities that the defender wants to protect from the adversaries.

However, due to the underlying network structure, the decision space of the interacting agents increases exponentially in the representation size of the game. This makes analyzing the game models almost intractable due to the limitation of computational resources. Despite the state of the art in hardware where the processing power is becoming faster, this is not sufficient to scale in exponential spaces where we have limited computational resources. In fact, the majority of people do not have access to the state of the art processing power which is also very expensive. A common way to utilize the best available resource is using a smaller, abstracted model of a larger model that can be analyzed within a feasible time duration. Doing so is not straightforward since we can lose overall quality. In this thesis, I particularly try to exploit the pattern, characteristics, and structure of the underlying network of a game-theoretic model to make algorithmic analysis scalable.

My first project is on Green Security Games that focuses particularly on the problem

of protecting wildlife and natural resources against illegal exploitation, such as poaching and illegal logging. The illegal activities can be prevented by patrolling the important areas using patrollers by following a particular route. The game model has an underlying network structure to capture the physical terrain. However, the graph representation entails a huge number of possible patrolling paths which grows exponentially with the size of the graph and it creates a computational challenge for the decision makers to find an optimal patrolling strategy. In this scenario, the poaching tends to happen more where there are high animal activities which create sparseness in the network. I present an algorithm that exploits the sparseness characteristic of the underlying network of the terrain to create a smaller representation which can be handled easily to compute the optimal strategy for the decision makers. The experiments show significant improvement over the base algorithms.

Next, I use a game-theoretic model based on a cyber defense scenario where a botnet spreads through the network and a network admin tries to increase the security of the network to stop that botnet. The network admin has limited resources. On top of that, a real-world network can have a huge number of machines which makes it difficult for the network admin computationally to allocate his limited resources for increasing the security of the network. However, most real-world networks are divided into subnets to increase performance and security. Botnets often spread easily within a subnet using worms that exploit existing vulnerabilities, but spreading between subnets is harder compared to intra-subnet due to existing security and monitoring. This locality leads to a game model with a particular structure that can be solved by decomposing the game into smaller games. I present an algorithm that utilizes this subnet structure in a network to achieve a highly scalable game model. The experiments show that using network decomposition the algorithm can give the best decisions within seconds.

Finally, I consider a cybersecurity game mode where an attacker tries to hide his identity and reach his goal node. He has some tool-sets and exploits in his possession which can overlap with other attacker types. The defender tries to pro-actively deploy deception to reveal more information about the attacker's identity. I show that the strategic use

of honeypots can reveal an attacker's identity earlier. However, in this scenario, both the defender's and the attacker's action space increases exponentially. To mitigate the scalability issue I reveal localized information of the attacker to the defender to help to reduce the action space. I also consider a real world network with Virtual Machines where I show that by analyzing sensor data the defender can get information on the attacker's location and reduce his action space to deploy honeypot strategically and dynamically to identify an attacker type earlier. This contribution is also relevant to the strategic use of resources rather than just fixed incident response from the defender's part which makes it harder for the attacker to pinpoint the defender's strategy.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Artificial Intelligence (AI) can be viewed as an invisible companion that is increasingly helping to make our daily life more convenient. In many areas of our life AI is involved with us; whether by helping us to find the best nearby restaurant or by giving us the best route to reach our destination; the examples are endless. However, by helping us AI is also making our lives more complex. We are faced with a problem of making decisions to achieve a goal in a huge environment where an environment can be either physical or virtual. We are interacting more with not only people but also intelligent artificial agents. The artificial intelligent agents are designed in such a way that they have a predefined objective and to achieve the goal the agent interacts with other agents and humans through a virtual model of the real world. Due to the size of action space available to choose from and a complex model of the situation, it is very difficult for a human to make the best decision. Game theory, a part of AI, analyzes the interactions between intelligent agents to make the best decisions for them.

Game theory models interaction among self-interested rational agents which is particularly useful in situations when there are limited resources available and the decision-making agent wants to optimize the allocation of the available resources to maximize his utility. For example, in the physical security domain, game theory is used to protect important structures, human lives, wildlife from the attacks of the adversaries. Recently there is a rise in the use of game theory in the cybersecurity domain too. In this dissertation, I particularly focus on game theoretic models where there is an underlying network structure involved where the agents interact. Typically a network has inter-connected nodes representing valuable entities that the defender wants to protect and the adversary wants to

compromise to maximize his gain. The interactions between agents in a network are modeled using different kinds of representations e.g. Normal Form Game (NFG), Extensive Form Games (EFG), Bayesian Game (BG), and Stackelberg Security Games (SSG). Many different solution concepts have been proposed for analyzing NFG and EFG including Nash equilibrium (NE). However, finding Nash equilibria[98] is known to be a computationally hard problem [55]. On top of that, the inherent structure of a network increases the size of interacting agents' strategy spaces exponentially which makes the game theoretic models unscalable, and finding a Nash Equilibrium computationally becomes a much harder problem. A common approach to tackle this scalability issue is reducing the strategy space by doing abstraction. Abstraction in game theory is a concept where the original game is shrunk using different techniques and the reduced game is then analyzed to compute the NE. After that, the solution is reversed back to the original game.

In this thesis, I work on different abstraction techniques to apply to different game-theoretic models with underlying network structure. Particularly, I will try to answer what attributes, characteristics, and structure of a network can be exploited to achieve a high-quality abstracted game model with high scalability in security games.

The first game theoretic model I present is Green Security Games (GSG). It's based on SSG and has an underlying network structure. GSG particularly focuses on the problem of protecting wildlife and natural resources against illegal exploitation, such as poaching and illegal logging. A grid-based graph is used to capture the physical terrain where each node represents a small area of the terrain and node utility is defined by animal activity in that small area. The defender who is in this case patrollers try to prevent poaching by patrolling the areas by following a particular route. The graph representation entails a huge number of possible patrolling paths and this leads to a major computational challenge because the possible number of patrolling paths grows exponentially with the size of the graph. One key observation is: in a graph representing an actual terrain the animal activity is sparse. Now my first research question is:

Q1. How can we exploit the sparsity of animal activity in the graph in GSG

to achieve an abstracted game model?

In GSG the poachers tend to poach more where there is high animal activity. Removing nodes with lower animal activity will produce a smaller game model. We use graph contraction to remove nodes with lower priority. However, we do not know the threshold values for the nodes to remove. In this work, we present a Double Oracle approach: strategy generation combined with graph contraction to automate the removal of lower priority nodes from the graph. Our experiments show approximately 99.6% improvements on scalability over the baseline algorithm with approximately 6% deviation from the optimal solution.

Next, I use a game-theoretic model based on a cyber defense scenario using an NFG for stopping the spread of an attacker (e.g., a botnet) in a network that has a subnet architecture. Most real-world networks are divided into subnets to increase performance and security, but there are limited resources to inspect/harden devices against cyber attacks. Automated Intrusion Detection Systems (NIDS/HIDS) is an essential defense, but it may not be possible to use a costly NIDS/HIDS on every network host. In this game model, the network administrator acts as the defender and a worm acts as the attacker. While NFG is a very general representation, it is often problematic to solve an NFG for real-world scenarios because enumerating all possible strategies results in an extremely large game. For example, in an enterprise network, the large number of hosts and interconnections can lead to intractable NFG models. My next research question is:

Q2. Can we use network decomposition utilizing the subnet structure to obtain a high-quality scalable NFG?

In a network, botnets often spread easily within a subnet using worms that exploit open ports and unpatched vulnerabilities. However, spreading between subnets requires moving through more secure and highly monitored routers that limit connectivity. This locality leads to a game model with a particular structure in a Normal Form Game (NFG). This particular structured NFG can be solved by decomposing the game into smaller subgames. I present an abstraction algorithm called Subgame Abstraction and Solution Concept (SASC)

which utilizes the partially independent subgame structures in an NFG to achieve a highly scalable game model. The experiments show that using network decomposition the SASC can solve huge NFG within seconds.

My next project focuses on validation by considering more realistic applications of a game theoretic model. I was asking myself, what is the point of scalability? Why do we want to make a game model scalable and what is the motivation behind it? What I found that one of the primary reasons behind wanting to make a game model scalable is the ability to use a game model in the real world. In the real world, computation time and response time matters. Otherwise, we wouldn't need the concept of scalability as long as the system or algorithm is feasible. So, my next project is about how we can transform a game theoretic model into an operational one rather than just show that it can compute a solution in case of a large input size. The idea of the project is a multi-agent adversarial interaction between an APT and a defender where the objective of the APT is to reach his goal machine by hiding his identity. The defender wants to reveal the identity of the APT and stop it by pro-actively using deception in a strategic fashion. I transform the game theoretic model into an empirical game theoretic model and then I take my first stab at handling the scalability issue of the empirical game theoretic model.

The project focuses on formally modeling what types of actions the defender can take to support more detailed APT identification early in the attack chain, allowing more information to target specific defensive responses. However, different APT may use the same core malware toolkits and common tactics. The APTs may intentionally try to look similar to other actors, and may even change during the course of an attack, leading to a complete change in focus. I constructed the game model in such a way that it's simple enough to start with and flexible enough to gradually add real-world complexities for future projects. The proposed game model is a multi-stage dynamic game that considers a post-exploitation scenario between an APT and the network defender. The game model is focused on detecting an APT early by deploying pro-active deception (honeypot/honey). However, when the defender wants to deploy deception he needs to evaluate all the honeypot deployment

settings in all the places possible in the network. As a result, the action space increases exponentially with the size of the network. Now the question is:

Q3.1. Can we exploit the locality of information and network security measures?

In this game model, if the defender has no information on the attacker's current position then he has to consider every possible action space to evaluate the deployments of different honeypots in different positions of a network. To mitigate the issue we reveal some information about the attacker's last position on the network to the defender. Our experiments show that information reveal can help to predict the next location of the attacker; and the defender can further reduce the action space to make the game model more scalable.

Next, I incorporated real-world constraints and rules in the game model to transform it into an empirical game theoretic model. For example, in a computer network, it is most probable that when an attacker just had a foothold of the network, he does not know about the topology of the whole network. Also, since defender can use moving target defense and dynamic honeypot so it is unrealistic to give accurate topology information to the attacker. So, unlike the game theoretic model, in this empirical model, the attacker cannot compute an attack path in the network. I had to consider how an attacker can make a plan to choose an optimal strategy that goes with his preference, so the model transformation was a big part of this project. However, in this project, the defender's action space increases with the uncertainty of the attacker's location. So, the question is:

Q3.2. How can a NIDS reduce the action space of the defender to make the algorithm work in a real-world network?

In the previous game-theoretic model we assumed that the defender knows the last position of the attacker. However, that is unrealistic in the sense that there is no perfect detection system to detect an attacker's position accurately. However, for this project sensing is the only way we can reduce the defender's action space effectively. The defender

needs to sense the environment using sensors which can be analyzed (using NIDS) to predict an attacker's position which can mitigate the problem of huge action space considered by the defender. From the sensor's data collection and utilizing the CDES detection module, we were able to reduce the action space of the defender. Finally, the experiments show that the defender can strategically deploy honeypot in the network to identify an attacker's identity.

For the next couple of chapters, I discuss related works and background. After that, I explore the research questions in different domains starting with the GSG.

Chapter 2

Related Work

2.1 Green Security Game

Green Security Game (GSG) [46, 69, 123] focuses on protecting wildlife and natural resources from environmental crimes such as poaching and illegal logging [57]. Limited resource is a common issue in the GSG domain because of the vast area that needs to be protected by patrolling. There are initial works on GSG game models [104, 119] which enumerates all possible combinations of the action spaces. Some improvements were made in the next iteration of the game models [75, 100]. Still, they are not efficient enough to handle a large scale GSG model. There are works in the SSG model that tries to handle large scale game model by using a more compact representation [79, 75]. Some works focus on solving the game models more efficiently by utilizing efficient optimization algorithms e.g. branch and bound, cutting plane [124, 75]. The most recent application called PAWS [56] approaches the scalability issue by incorporating cutting plane and column generation techniques.

2.2 Game Theory for Cyber Domain

Game theory in cybersecurity is a particularly evolving and challenging domain. Most of the current approaches in the cyber domain use adhoc defense mechanisms or ML techniques. These models ignore a crucial part of the cyber domain that is the multi-agent adversarial interactions. However, due to the complexities, scalability issues, and learning curve of the game theoretic models it did not get popularity in the cyber industry even though it

is very promising. The research area is however continuing to flourish with contributions to the theoretical aspect of game theoretic models in the cyber domain. A survey by Roy et al. [109] listed comprehensive works of game theory in the cyber domain. Jormokka et al. [77] used the static game with imperfect information to represent cyber warfare game. Carin et al. [49] introduced a static game model for the strategic protection of intellectual properties in the cyber domain. Liu et al. [92] presented a game theoretic model to represent the interaction between a DDoS attacker and a network admin. Liu et al. [93] proposed a Bayesian game model to solve network intrusion detection problem by updating the belief on an attacker's actions. Chen [51] introduced a game model that minimizes the worm propagation speed in a network using a dynamic game. Alpacan et al. [37] used a Markov game model to represent adversarial interactions between IDS and an attacker where the attacker can have imperfect information about the sensor of the IDS. Bloem et al. [42] modeled the intrusion response as a resource allocation problem. Alpacan et al. [36] modeled the interaction between an attacker and a network admin using repeated games where the sensor is imperfect to detect an attacker.

2.3 Game Theory for Cyber Deception

Game theory holds its special place in cyber deception where the strategic decision is of utmost importance to maximize strategic randomness and to improve robustness, resource allocation, and resiliency. Radek et al. [105] proposed a game model for effective honeypot allocation in the network. Kiekintveld [80] discussed several game models for the strategic use of honeypots in the network. Horak et al. [72] introduced a game model to pro-actively manipulate adversary's behavior to thwart an attacker from learning the target network. Jajoda et al. [76] describe several works in cyber deception where game theory can be utilized vastly. Among recent works Zhu et al. [126] introduced a tutorial and taxonomy on the use of cyber deception using incomplete information, dynamic games, mechanism design theory.

2.4 Abstraction

To analyze games that are beyond the limits of standard solution algorithms, an increasingly common approach is to apply some form of automated abstraction to simplify the game. The simplified game is then analyzed using an available solver, and the solution is somehow mapped back into the original game. If the simplified game can retain the key strategic features of the original game, then in principle the solution of the simpler game may be a reasonable approximation of the solution to the original game. This general approach has been very successful in developing computer poker agents, and most of the successful players in the annual competition in computer poker over the past years have used some variation of this idea (e.g., [48, 44, 62, 63, 65, 66, 127]).

Many of the recent works on abstraction focus on extensive form games with sequential interactions and uncertainty, in part due to the motivation of poker agents. These include a wide variety of specific methods including both lossless abstractions [64] and lossy abstractions [110]. While many lossy methods do not provide bounds on the error introduced by the abstraction, some recent work has been able to provide theoretical bounds for very general classes of games [86, 85]. Another recent example considers imperfect recall abstractions with earth mover's distance [59] for a hierarchical abstraction [47] technique.

State of the art techniques particularly focus on computing a NE in an iterative fashion. Two types of algorithms have been used heavily: regret minimization algorithm based on *counterfactual regret minimization* [127, 89, 44] (CFR) and *first-order methods* (FOMs) e.g. excessive gap techniques (EGT) [99] with proper *distance generating functions* (DGF) for EFG strategies [70, 87, 83, 88].

The approach behind CFR is to decompose overall regret into a set of additive regrets and minimizing them independently. Minimizing the immediate regrets minimizes the overall regrets and after playing a two-player zero-sum game repeatedly NE is reached eventually. To handle huge games with imperfect-information abstraction technique is used by bucketing similar states together. Then the simplified game is approximately

solved using tabular CFR. A variant of CFR called CFR^+ [117] was used to near optimally solve heads-up limit Texas Hold'em [44].

Another variant of CFR called MCCFR [89] was used to make computer poker agents called *Libratus* by solving subgames [48] which beat expert human poker players in no-limit Texas Hold'em which has 10^{161} decision points before abstractions. This poker agent has three main modules. In the first module, an abstracted game was generated by using action abstraction and card abstraction. In the action abstraction, less number of bets are used by using granular bettings. For the card abstraction, similar hands are put together and they are treated identically. Then MCCFR is used to generate an approximate strategy for the whole game. The solution provided a detailed strategy for the earlier parts of the game but a less fine grained strategy for the latter rounds of the game. When the game reaches the later portion of the game, the second module of *Libratus* builds a fine grained abstracted model of the subgame and solves the subgame in real-time. The third module fills in the missing pieces of the abstracted game by using the actions of the opponent and creates a strategy for those parts. There is also another computer poker agent called *DeepStack* [97] that uses action abstraction and CFR to solve Heads Up No limit Poker.

In another work [84] a FOM called EGT with dilated entropy distance function can compete with CFR techniques to solve large scale sequential games. For experiments the authors used EGT to solve the subgames in the *Libratus* agent to solve no-limit Texas Holdem Poker. The results show that EGT can outperform CFR in many cases. There are works in the literature on abstraction techniques on game models related to GSG (e.g. EFG, SSG) [86, 78, 82] that focus on providing a unified framework to abstract the EFG action space to approximate Stackelberg Equilibriums.

Chapter 3

Background

3.1 Normal Form Games

Game theory analyzes interactions between multiple intelligent players by modeling the scenarios as a game. A game can be either one shot, sequential or repetitive. Normal Form Game (NFG) is the most popular form of game to represent player interactions and strategy space. There is also Stackelberg Security Game, Green Security Game (GSG) and Extensive Form Game (EFG).

A normal form game captures all possible combinations of strategies for the players in a matrix form. A player can choose either a single "pure" strategy or play a "mixed" strategy that specifies a probability distribution over the pure strategies. The goal for all players is to maximize their expected utility. Formally, a finite, N-person normal-form game is described by a tuple (N, A, u), where [114]:

- N is a finite set of N players, indexed by i.
- $A = A_1 \times ... \times A_n$, where A_i is a finite set of actions available to player i. Each vector $a = (a_1, ..., a_n) \in A$ is called an action profile. So, $a_i \in A_i$ is pure action in the original game. $a_{i,k}$ is th kth action for player i.
- $s_i \in S_i$ is the space of mixed strategies in the game.
- $u = (u_1, ..., u_n)$ where $u_i : A \mapsto R$ is a real-valued utility (or payoff) function for player i, which is extended to mixed strategies as usual by using expected utility.
- $\pi^i(a_i)$ gives the probability of action a_i for player i.

- $A_i(O)$ gives the set of available actions for player i in NFG O.

Throughout this paper, we will consider abstractions that are also represented as (simpler) normal-form games. We will use the following modified notation to refer to an abstracted game $(\hat{N}, \hat{A}, \hat{u})$:

- $\hat{A} = \hat{A}_1 \times ... \times \hat{A}_n$, where $\hat{A}_i$ is a finite set of actions available to player i, with action profiles $\hat{a} = (\hat{a}_1, ..., \hat{a}_n) \in \hat{A}$. We will use $\hat{a}_{i,k}$ for the kth action for player i.
- $\hat{u} = (\hat{u}_1, ..., \hat{u}_n)$ where $\hat{u}_i : \hat{A} \mapsto \hat{R}$ is a real-valued utility (or payoff) function for player i.
- $\hat{a}_i \in \hat{A}_i$ is a pure action in the abstracted game.
- $\hat{s}_i \in \hat{S}_i$ is a mixed strategy in the abstracted game.
- $\hat{A}_i(R)$ is the set of available actions for player i.
- $\pi^i(\hat{a}_i)$ gives the probability of action $\hat{a}_i$ for player i.

Table 3.1 is an example of a two-player normal form game.

	C	D
A	4,4	6,2
B	1,7	5,2

Table 3.1: Example Normal Form Game

The row player can choose action A or action B. The column player can choose either action C or action D. If row player chooses action A and column player chooses action D, payoffs are 6 and 2 for row and column player respectively. Main purpose of modeling an interaction between players is to analyze the game model and find the strategy for the players where every player acts rationally and maximizes his own outcome. A strategy for a player can either be a pure strategy or a mixed strategy.

Definition 3.1 (Mixed Strategy[91]). Let (N, A, u) be a normal-form game and for any set X let $\Pi(X)$ be the set of all probability distributions over X. Then the set of mixed strategies for player i is $S_i = \Pi(A_i)$.

In a pure strategy, only one action has positive support. And we want to find how the players choose their strategy to maximize their output. In a sense, every player wants to play his best against his opponent's strategy.

Definition 3.2 (Best Response[91]). Player i's best response to other players ($-i$)'s strategy profile s_{-i} is a mixed strategy $s_i^* \in S_i$ such that $u_i(s_i^*, s_{-i}) \geq u_i(s_i, s_{-i})$ for all strategies $s_i \in S_i$.

When all players are performing the best response to all other players this is known as a Nash Equilibrium (NE). This is described more formally in Definition 3.3.

Definition 3.3 (Nash Equilibrium[91]). A strategy profile $s = (s_1, \ldots, s_n)$ is a Nash equilibrium if, for all agents i, s_i is a best response to s_{-i}.

The outcome of the example game is a pure strategy Nash equilibrium because all players are playing the best response to all other players.

3.1.1 Solution Concepts

Once the abstracted game is constructed, we must choose a solution concept to analyze this simplified game. A natural choice that is often used is to search for a (possibly approximate) Nash equilibrium of the abstracted game. We consider both pure and mixed-strategy Nash equilibrium, as well as a different concept that directly minimizes a bound on the approximation quality in the original game.

Approximate Pure Strategy Nash Equilibrium

In a pure-strategy Nash equilibrium (PSNE) all players play pure strategies that are mutual best-responses, so no player can gain by changing to a different pure strategy. It is simple

to check whether a given outcome is a PSNE by simply checking the possible gains from deviation. However, PSNE are not guaranteed to exist. Therefore, we instead look for the pure-strategy outcome the is the best approximate equilibrium, with the minimum possible gain for any player to change to a different strategy. This is known as an ε-Nash equilibrium (in a Nash equilibrium $\varepsilon = 0$). We first calculate the values of deviations for each action a_m using equation 3.1, and then select the action profile that minimizes the maximum benefit to deviation, as in equation 3.2

$$\varepsilon(a_i^* \in A_i) = max_{\forall a_i \in A_i, \forall a_j \in A_j}[u_i(a_i, a_j) - u_i(a_i^*, a_j)] \quad \forall j \in N, i \neq j \tag{3.1}$$

$$(a_i \in A_i) = min_{\forall a_i \in A_i}[\varepsilon(a_i)] \qquad \forall i \in N \tag{3.2}$$

Mixed Strategy Nash Equilibrium

We also calculate a version of mixed-strategy Nash equilibrium using the software package Gambit [95]. There are several different solvers for finding Nash equilibria in this toolkit. We used one based on Quantal response equilibrium (QRE) [67] that is based on a tracing procedure with noisy best-response functions for the players. In this limit as the noise in the best responses goes to zero this method converges to a Nash equilibrium. In practice, we have found this to be a relatively reliable way to find a sample Nash equilibrium.

3.2 Stackelberg Security Game

Stackelberg Security Game can modeled as a NFG with two players, a defender, Θ, and an attacker, Ψ, where each player can be either an individual or a group. Each player has a set of possible pure strategies $\sigma_\Theta \in \Sigma_\Theta$ and $\sigma_\Psi \in \Sigma_\Psi$. Each player also has a mixed strategy, $\delta_\Theta \in \Delta_\Theta$ and $\delta_\Psi \in \Delta_\Psi$, played over the pure strategies using a probability distribution. Payoffs for defender are defined for joint pure strategy outcomes as $\Omega_\Theta : \Sigma_\Theta \times \Sigma_\Psi \Rightarrow R$

and for attacker $\Omega_\Psi : \Sigma_\Theta \times \Sigma_\Psi \Rightarrow R$. Given a mixed strategy we can easily compute the payoff by taking the expected value over the pure strategy payoffs.

In an SSG there is always a leader and a follower. Leader commits over a strategy. Follower observes leader strategy and responds with his best strategy. The SSG model allows us to model a scenario where defender acts as a leader. Adversary acts as follower where they can put surveillance to monitor defender strategy and act depending on the observations. In a SSG attacker selects a best response after observing defender's action which can be formulated as $F_\psi : \Delta_\Theta \Rightarrow \Delta_\Psi$.

3.2.1 Stackelberg Equilibrium

Standard solution concept in a game model is Nash Equilibrium where no one can gain by unilaterally deviating to another strategy, that means every player plays the best response to each other. Stackelberg Equilibrum in an SSG is a refinement of the Nash Equilibrium profile where each player plays according to subgame perfect equilibrium in a subgame of the original game. The subgame perfect equilibrium eliminates any form of threats which has no form of credibility. There are two kinds of Stackelberg Equilibrium: strong and weak. In a Strong Stackelberg Equilibrium (SSE) attacker plays for defender's optimal payoff in case there is a tie between strategies. In a Weak Stackelberg Equilibrium follower plays the worst strategy for the defender.

Definition 3.4 (Strong Stackelberg Equilibrium[90, 45])**.** A pair of strategies (δ_θ, F_ψ) form an SSE if they satisfy the following conditions:

1. The leader plays best response : $\Omega_\Theta(\delta_\Theta, F(\delta_\Theta)) \geq \Omega_\Theta(\delta'_\Theta, F(\delta'_\Theta)) \quad \forall \delta'_\Theta \in \Delta_\Theta$
2. The follower plays best response : $\Omega_\Psi(\delta_\Theta, F(\delta_\Theta)) \geq \Omega_\Psi(\delta'_\Theta, \delta_\Psi) \quad \forall \delta'_\Theta \in \Delta_\Theta, \forall \delta'_\Psi \in \Delta_\Psi$
3. The follower optimally breaks ties for the leader : $\Omega_\Psi(\delta_\Theta, F(\delta_\Theta)) \geq \Omega_\Psi(\delta'_\Theta, \delta_\Psi) \quad \forall \delta'_\Theta \in \Delta_\Theta, \forall \delta'_\Psi \in \Delta^*_\Psi$, where Δ^*_Ψ is the set of best responses of the follower.

	Covered	Uncovered
Defender	5	-20
Attacker	10	30

Table 3.2: Compact representation for an attacked target

In a SSG playing a deterministic strategy by the leader is a guaranteed loss because the follower will be able to exploit that. But a planned randomized strategy to maximize the payoff is always an advantage.

3.2.2 Compact Security Game Model

In an SSE the defender tries to protect valuable targets, airports, banks, ports, by allocating resources. In a normal form representation sometimes the number of targets are high and the problem space suffers from combinatorial explosion. To avoid that there is a compact representation of the SSG. Table 3.2 shows the compact representation of a game for an attacked target. The important feature of this model is that the payoff depends only on the attacked target and whether the target is covered or not. From payoffs perspective all resource allocations are identical. We have a coverage vector C, which gives us the probability of defending a target. We also have an attack vector A which gives us the probability of a target being attacked by the attacker.

Chapter 4

Abstraction in Green Security Games

4.1 Introduction

We face many complex security threats with the need to protect people, infrastructure, computer systems, and natural resources from criminal and terrorist activity. A common challenge in these security domains is making the best use of limited resources to improve security against intelligent, motivated attackers. The area of *green security* focuses on problems related to protecting wildlife and natural resources against illegal exploitation, such as poaching and illegal logging. Resource limitations are particularly acute in fighting many types of environmental crime, due to a combination of limited budgets and massive areas that need surveillance and protection. For example, it is common for small numbers of rangers, local police, and volunteers to patrol protected national parks that may cover thousands of square miles of rugged terrain [71].

Work on *green security games* [46, 57] has proposed formulating the problem of finding optimal patrols to prevent environmental crime as a Stackelberg security game [79]. In these games, the defender (e.g., park ranger service) must decide on a randomized strategy for patrolling the protected area, limited by the geographic constraints and the number of available resources. The attacker (e.g., poacher) selects an area of the park to attack based on the intended target and knowledge of the typical patrolling strategy (e.g., from previous observations and experience). Green security games are used to find randomized patrolling strategies that maximize environmental protection given the resources available.

Green security games typically model the movement constraints for the defender patrols using a graph representing the physical terrain. Unfortunately, this leads to a major

computational challenge because the number of possible paths for the defender grows exponentially with the size of the graph. Enumerating all possible combinations of paths for multiple resources makes the problem even more intractable [104, 119]. Several algorithms have been proposed in the literature to solve these games more efficiently [75, 100]. Most of these rely on incremental strategy generation (known as double oracle algorithms, or column/constraint generation) to solve an integer programming formulation of the problem without enumerating the full strategy space. The most recent application called PAWS [56] approaches the scalability issue by incorporating cutting plane and column generation techniques.

Here, we take a new approach that *combines* strategy generation methods with automated game abstraction methods based on graph contraction. The idea of using automated abstraction has been very successful in solving other types of very large games, such as computer poker [62, 63, 65, 66, 127]. The basic idea of our game abstraction is motivated by graph contraction techniques used to speed up pathfinding and other computations on graphs. When we apply graph contraction to a green security game, it dramatically reduces the strategy space for the defender, leading to lower solving time. To improve scalability even further we integrate graph contraction with strategy generation to create a new class of algorithms capable of solving very large green security games. We evaluate our new algorithms on graph-based security games motivated by the problems encountered in green security domains, including some based on real world data sets. The experiments show that we can dramatically improve solution times by using abstraction in combination with strategy generation, leading to high-quality approximations within seconds even for graphs with a thousand nodes.

4.2 Related Work

The first approach to compute security resource allocations was to find a randomized strategy after enumerating all possible resource allocations [104], which is used by the Los

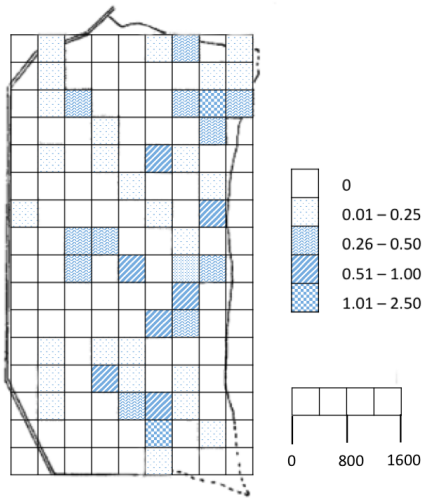
Angeles Airport Police in an application called ARMOR [106]. A more compact form of security game representation was used [79] to develop a faster algorithm (IRIS [119]), which is used for scheduling by the Federal Marshal Service (FAMS). ASPEN [75] was introduced to deal with the exponential size of games with complex scheduling constraints by using a branch-and-price approach. Most recently, to tackle more massive games an approach based on cutting planes was introduced [124] to make the solution space more manageable. Game theoretic algorithms are also used to secure ports [112] and trains [125]. Recently, successful deployment of game theoretic applications motivated researchers to use game theory in green security domains [46, 69, 123]. This led to a new game model called GSG [57]. Assumptions about the attacker being able to fully observe the defender strategy can be unrealistic in some cases, so partial observability and bounded rationality have been introduced to make the attacker model better fit the practice. Defender payoff uncertainty has also been addressed with these issues in an algorithm called ARROW [100]. Despite the models and algorithms introduced, how to handle the large strategy space in GSGs remains a challenge. In this paper, we introduce abstraction techniques to address this problem. Many abstraction techniques have been developed for extensive form games with uncertainty including both lossy [110] and lossless [64] abstraction. There has been some work which gives bounds on the error introduced by abstraction [85]. There are also imperfect recall abstractions that consider hierarchical abstraction [47] and Earth mover's distance [59].

Graph contraction techniques [60] have been used to achieve fast routing in road networks, where contraction acts as a pre-processing step. This method has been improved using fast bidirectional Dijkstra searches [115, 61]. A time-dependent contraction algorithm has also been introduced for time-dependent road networks [41]. Graph contraction has also been used in imperfect information security games with infinite horizon where the area is patrolled by a single robot [40]. In this paper, we leverage insights from graph contraction to handle the large strategy space in GSGs. Another recent closely related work [74] uses cut-based graph contraction and also column generation approach for restricting the

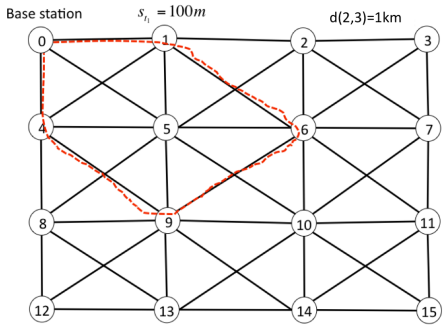
strategy space, but for a different type of security model based on checkpoint placement for urban networks.

4.3 Domain Motivation

Illegal activities such as poaching pose a major threat to biodiversity across all types of habitats and many species such as rhinos and tigers. A report [33] from the Wildlife Conservation Society (WCS) on May 2015 stated that the elephant population in Mozambique has shrunk from $20,000$ to $10,300$ over the last five years. Elephants were recently added to the IUCN Red List [34]. Marine species also face danger due to illegal fishing and overfishing, causing harm to the people of coastal areas who depend on fishing for both sustenance and livelihood. According to World Wide Fund for Nature (WWF), the global estimated financial loss due to illegal fishing is $23.5 billion [35]. Organizations like WCS are studying strategies for combating environmental crime that include patrols of both land and sea habitats to detect and deter poaching. PAWS [56] is a new application based on green security games that help to design patrolling strategies to protect wildlife in threatened areas. The area of interest is divided into grid cells that capture information about the terrain, animal density, etc. Each grid cell is a potential target for the poachers. The patroller plans a route to protect the targets along a path. However, if the grid cell is too large (e.g., 1km by 1km) or the terrain is complex, it is very difficult for the patroller to patrol even a single grid cell without any detailed path provided in the cell. Therefore, a fine-grained discretization is often required, leading to a large number of targets and an exponential number of patrol routes that existing solvers cannot handle. PAWS handles this problem by pre-defining a limited set of routes based on domain knowledge of features like ridgelines and streams, which can be found based on elevation changes. We also observe that in many green security domains, there is a high variance in the importance of the targets. For example, Figure 4.1a shows the mean number of elephants in each area of a grid representing the Queen Elizabeth National Park in Uganda [58]. There are many cells that



(a) Mean numbers of elephants/$0.16km^2$ in Queen Elizabeth National Park, Uganda



(b) A graph representation of a grid-based GSG (a patrolling path is shown in red).

Figure 4.1: Domain example and game model.

have no animal count at all, and if there is minimal activity it is very inefficient to consider these areas as targets to patrol (or poach). This motivates our abstraction-based approach to make it computationally feasible to directly analyze high-fidelity maps for green security without preprocessing.

4.4 Game Model and Basic Solution Technique

A typical green security game (GSG) model is specified by dividing a protected wildlife area into grid based cells, as shown in Figure 4.1a. Each cell is considered a potential target t_i where an attacker could attempt a poaching action. We transform this grid-based representation into a graph as shown in Figure 4.1b. Each node represents a target t_i.

Definition 4.1 (GSG Graph)**.** A GSG Graph is a graph $G = (V, E)$ where each node $t_i \in V$ is associated with a patrolling distance s_{t_i} and each edge $e_{ij} \in E$ is associated with a traveling distance $d(i, j)$. There exists a base node $B \in V$. A feasible patrolling path is a sequence of consecutive nodes that starts and ends with B, with a total distance that does

not exceed the distance limit d_{max}.

For example, in Figure 4.1b, $s_{t_1} = 100m$. This means that to protect target t_1, the patroller needs to patrol for a distance $100m$ within target t_1. $d(2,3) = 1km$ indicates the distance from target t_2 to t_3. The defender patrols to protect every target on the patrolling path. Therefore, the total distance of a path is the sum of patrolling and travel distance. Typically the patrol starts in a base station and ends in the same base station. For example, a patrolling path is shown in Figure 4.1b where the patrol starts at t_0 and traverses through targets $t_1 \rightarrow t_6 \rightarrow t_9 \rightarrow t_4$ and ends back in target t_0.

The defender has a limited number of resources R, each of which can be assigned to at most one patrolling path that covers a set of targets $t \in T$. So the defender's pure strategies are the set of joint patrolling paths $J_m \in J$. Each joint patrolling path J_m assigns each resource to a specific path. We denote a patrolling path by p_k and the base target by t_b. The length of p_k is constrained by d_{max}.

We use a matrix $P = P_{J_m t} = (0,1)^n$ to represent the mapping between joint patrolling paths and the targets covered by these paths, where $P_{J_m t}$ represents whether target t is covered by the joint patrolling path J_m. We define the defender's mixed strategy x as a probability distribution over the joint patrolling paths J where x_m is the probability of patrolling a joint patrolling path J_m. The coverage probability for each target is $c_t = \sum_{J_m} P_{J_m t} x_m$.

If target t is protected then the defender receives reward $U_d^c(t)$ when the attacker attacks target t, otherwise a penalty $U_d^u(t)$ is given. The attacker receives reward $U_a^u(t)$ if the attack is on an area where the defender is not patrolling, or penalty $U_a^c(t)$ if the attack is executed in a patrolled area. These values can be based on the density of the animals in the area attacked, as a proxy for the expected losses due to poaching activities. We focus on the zero-sum game case where $U_d^c(t) = U_a^c(t) = 0$ and $U_d^u(t) = -U_a^u(t)$. In the rest of the paper, we also refer to $U_a^u(t)$ as the utility of target t.

We use the Stackelberg model for GSG. In this model, the patroller, who acts as defender, moves first and the adversary observes the defender's mixed strategy and chooses a

strategy afterward. The defender tries to protect targets $T = t_1, t_2, ..., t_n$ from the attackers by allocating R resources. The attacker attacks one of the T targets. We focus on the case where the attacker is perfectly rational and compute the Strong Stackelberg Equilibrium (SSE) [90, 45, 121], where the defender selects a mixed strategy (in this case a probability distribution x over joint patrolling paths J_m), assuming that the adversary will be able to observe the defender's strategy and will choose the best response, breaking ties in favor of the defender. Given a defender's mixed strategy x and the corresponding coverage vector c, the expected payoff for the attacker is

$$U_a(c, t) = \max_{t \in T} \{(1 - c_t) U_a^u(t)\} \tag{4.1}$$

It is possible to solve this problem by enumerating all feasible joint patrolling paths [75]. In the case of zero-sum games, the optimal patrolling strategy for the defender can be determined by solving the following linear program (LP).

$$\min_{x,k} \quad k \tag{4.2}$$

$$(1 - Px)U_a^u \leq k \tag{4.3}$$

$$\sum\nolimits_i x_i \leq 1 \tag{4.4}$$

$$x \geq 0 \tag{4.5}$$

Equation 4.2 represents the objective function, which minimizes the expected payoff for the attacker, or equivalently, maximizes the expected payoff for the defender. Constraint 4.4 makes sure that the probability distribution over the joint patrolling paths does not exceed one. The solution of the LP is a probability distribution x over the joint patrolling paths J, and this is the strategy the defender commits to. The attacker will choose the target with highest expected utility, as shown in Constraints 4.3. This formulation does not scale well to large games due to the exponential number of possible joint paths as the graph grows larger.

4.5 Solving GSG with Abstraction

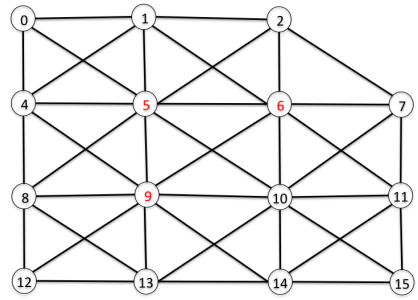
We describe our approach in three stages. First, we describe our method for contracting a graph by removing nodes and calculating a new set of edges to connect these nodes that retain the shortest path information. This contracted graph can be solved using any existing algorithm for GSG; as a baseline, we use the LP on the full set of paths. Second, we describe a single-oracle approach for finding the set of targets that must be included in the contracted game. This method restricts the set of targets to a small set of the highest-valued targets, and iteratively adds in additional targets as needed. Finally, we describe the double-oracle algorithm This uses the same structure as the single oracle, but instead of solving each restricted game optimally, we restrict the defender's strategy space and use heuristic oracles to iteratively generate paths to add to the restricted game.

4.5.1 Graph Contraction

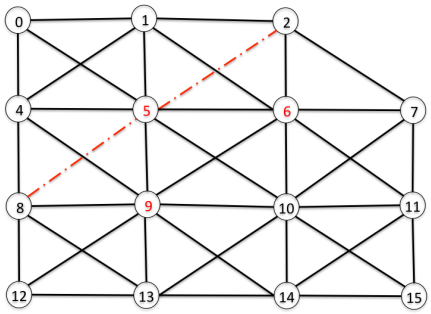
Our approach combines the key ideas in double oracle methods and graph contraction. There are often relatively few important targets in a GSG. For example, the key regions of high animal density are relatively few, and many areas have low density, as shown in Figure 4.1a. This suggests that many targets in the game can be removed to simplify the analysis while retaining the important features of the game.

We first describe how we construct an abstracted (simplified) graph for a restricted set of target nodes. Essentially, we remove all of the nodes except the restricted set, and then add additional edges to make sure the shortest paths are preserved.

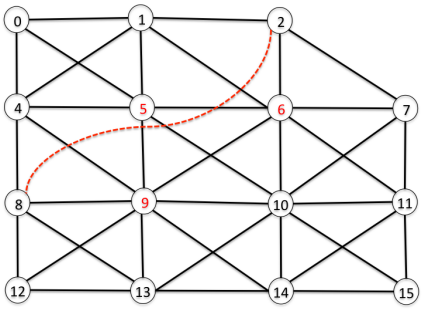
Many graph contraction procedures used in pathfinding remove nodes one by one, but we use a contraction procedure that removes the nodes in one step. Suppose we have decided to remove the set of nodes $T_u \in T$. We find all the neighbors of set T_u, denoted as V. Next we try to find the shortest paths between each pair of nodes $(v_i, v_j) \in V$ that traverse through nodes T_u where v_i and v_j are not adjacent. We use Floyd-Warshall algorithm [53] to find the shortest paths for all the nodes in V using only nodes T_u. If the



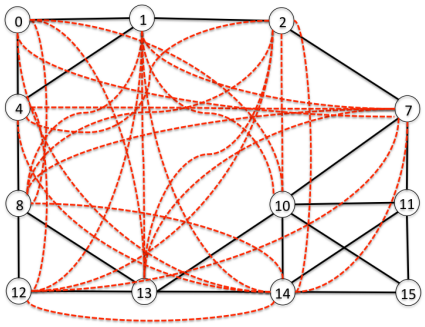
(a) Unnecessary nodes $5, 6, 9$



(b) Edge $8- > 5- > 2$ to be removed



(c) New shortcut path $8- > 2$



(d) Final graph after contraction of node $5, 6, 9$

Figure 4.2: Instant Contraction procedure for different nodes

length of the shortest path does not exceed d_{max}, we add an edge (v_i, v_j) in the contracted graph, with distance equals the length of the shortest path.

Figure 4.2 shows how the contraction works. Figure 4.2a shows the removed nodes $T_u = (5, 6, 9)$. The neighbor set of T_u is $V = (0, 1, 2, 4, 7, 8, 10, 12, 13, 14)$. For convenience, we show a breakdown of the step in Figure 4.2b where the edge $(8 \rightarrow 5 \rightarrow 2)$ is shown and in Figure 4.2c where the edge $(8 \rightarrow 5 \rightarrow 2)$ is replaced with shortcut $8 \rightarrow 2$. Figure 4.2d shows the final stage of the graph after contracting nodes $5, 6, 9$. Algorithm 1 shows pseudocode for the contraction procedure.

Reverse Mapping

When we solve a GSG with a contracted graph (e.g., using the standard LP), the paths found in the solution must be mapped back to the paths in the original graph so they

Algorithm 1 Instant Contraction Procedure

1: **procedure** InstantContractGraph ▷
2: $G \leftarrow Graph()$ ▷ Initiate the graph to contract
3: $n_d \leftarrow ContractedNodes()$ ▷ Get the nodes to contract
4: $n_{nei} \leftarrow ComputeNeighbors(n_d)$
5: $apsp \leftarrow AllPairShortestPath(G, n_d, paths)$
6: **for** $v \leftarrow neighbors.pop()$ **do**
7: **for** $v' \leftarrow neighbors.pop()$ **do**
8: **if** $v \neq v' \& notadjacent(v, v')$ **then**
9: $d \leftarrow apsp[v][v']$
10: $path \leftarrow getPath(paths, v, v')$
11: **if** $d \leq dmax$ **then** ▷ if d is less than the distance limit
12: $UpdateNeighbors(v, v', path, d)$
13: $v.AddNeighbor(v', path)$
14: $v'.AddNeighbor(v, path)$
15: **end if**
16: **end if**
17: **end for**
18: **end for**
19: $RemoveAllContractedNodes(G, n_d)$
20: **end procedure**

can be executed. This is because a single edge in the abstract path can correspond to a path of several nodes in the original graph. In algorithm 1, when the contracted graph is constructed, the corresponding path in the original graph of each edge being added is already recorded, and is the basis the reverse mapping.

Theorem 4.2. *The contraction process described in Algorithm 1 preserves the shortest paths for any pair of nodes that are not removed in the original graph. Formally, given a graph $G = (T, E)$ and a subset of nodes T_u, Algorithm 1 provides a contracted graph $G' = (T \setminus T_u, E')$ and the length of the shortest path for any pair of nodes $(v_i, v_j) \in T \setminus T_u$ in G' is the same as in G.*

Proof. Let $t_1 \to t_2 \to t_3 \to \ldots \to t_i \to \ldots \to t_j \to \ldots \to t_k$ is the shortest path between t_1 and t_k and $d(G, t_1, t_k) = x, d(G', t_1, t_k) = x'$ where $x < x'$ and $t_i, ..., t_j \in T_u$ nodes are being contracted. So,

$$
\begin{aligned}
d(G', t_1, t_k) &= x' \\
&= t_1 \to t_2 \to t_3 \to \ldots \to t_k \\
&= t_1 \to t_2 \to t_3 \to \ldots \to t_k \\
&= t_1 \to t_2 \to t_3 \to ... \to t'_i \to \ldots \to t'_j \to ... \to t_k && \text{(Reverse mapping)} \\
&= t_1 \to t_2 \to t_3 \to ... \to t_i \to \ldots \to t_j \to ... \to t_k && \text{(Algorithm 1)} \\
&= x
\end{aligned}
$$

□

So, out initial assumption of $x < x'$ was wrong.

4.5.2 Single-Oracle Algorithm Using Abstraction

We begin by describing a basic "single oracle" algorithm that restricts only the attacker's strategy space (i.e., the number of targets). The basic observation that leads to this approach is based on the notion of an attack set. In the Stackelberg equilibrium solution

Algorithm 2 Single Oracle With Abstraction (SO)

Input: original graph G, target utility U_i, $\forall i \in V$

Output: defender mixed strategy x and coverage vector c

1: $\bar{T}$=GreedyCoverR(G) ▷ Find initial set of targets to be considered in the restricted graph
2: Set current graph $G_c = G$
3: **repeat**
4: G_t =`Contract`$(G_c, \bar{T})$ ▷ Contract graph
5: $(u, x_t, c_t) = Solve(G_t)$ ▷ Solve restricted graph, get attacker's expected utility u, defender strategy x_t, coverage vector c_t
6: $v = AttEU(G_c, c_t)$ ▷ Calculate actual attacker's expected utility on current graph
7: **if** $v == u$ **then**
8: Break
9: **end if**
10: G_c =`ContractWithThreshold`(G_c, u) ▷ Remove targets with utility $< u$
11: **if** G_c is small enough **then**
12: $(u, x, c) = Solve(G_c)$ ▷ Solve G_c directly
13: Break
14: **end if**
15: Add at least one additional target into $\bar{T}$
16: **until** $1 < 0$

to a security game, there is a set of targets that the attacker is willing to attack; this is the set that the defender must cover with positive probability. All other target have maximum payoffs lower than the expected payoff for the attacker in the equilibrium solution, so the attacker will never prefer to attack one of these targets, even though it is left unprotected. If we could determine ahead of time which set of targets must be covered in the solution, we could simply apply graph contraction to this set of targets, solve the resulting game,

and be guaranteed to find the optimal solution.

Our approach is to start by considering only a small set of targets $\bar{T}$, perform contraction, and solve the abstracted game for this small set of targets. If the attacker expected value in the solution is lower than the value the attacker can get from attacking the best target that was not included in the restricted game, we add at least one (and possible more than one) additional target to the restricted game and repeat the process. Targets are added in decreasing order of the attacker's payoff for attacking the target if it is not protected at all. If we solve a restricted game and the attacker's expected value is greater than the unprotected values of all remaining targets, we can terminate having found the correct attack set and the optimal solution.

The initial set of targets to be considered is determined by *GreedyCoverR* (GCR). First consider the case where there is only one patroller. We use an algorithm named GC1 to find a greedy patrolling path. GC1 greedily inserts targets to the path and asks the patroller to take the shortest path to move from one target to the next target. The targets are added sequentially in a descending order of the target utility. GC1 terminates when the distance limit constraint is violated. GCR calls GC1 R times to find greedy paths for R patrolling resources. If the greedy paths can cover the top K targets, GCR returns the set of targets whose utility is no less than the utility of the $(K+1)^{th}$ target. This is because a restricted graph with the top K targets can be perfectly protected given the greedy paths, and therefore the patroller can try to protect more targets.

Algorithm 2 shows psuedocode for this procedure. Clearly, u is non-decreasing and v is non-increasing with each iteration. For a value of u in any iteration, we can claim that any target whose utility is smaller than u can be safely removed as those targets will never be attacked (attacker will not deviate if those targets are added to the small graph). The function `Contract`$(G, \bar{T})$ completes two tasks. First, it removes targets that are not in $\bar{T}$, and second, refine the graph by removing dominated targets. In each iteration, u provides a lower bound of the attacker's expected utility in the optimal solution (optimal defender strategy) and v provides an upper bound. If $v == u$, it means current solution is the

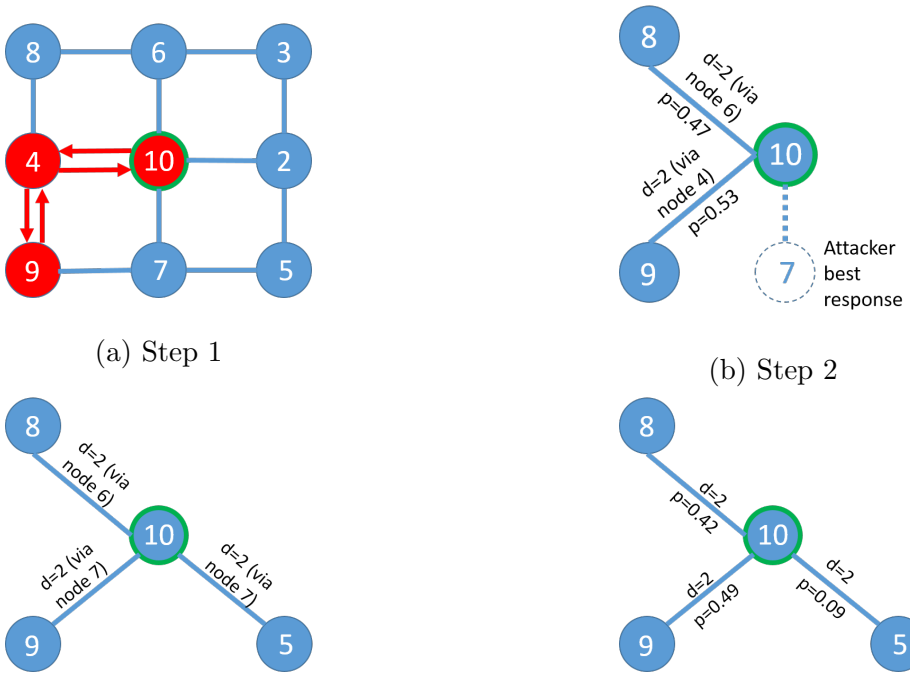
optimal. Line 15 adds at least one target to the set $\bar{T}$. Figure 4.3 illustrates the algorithm on an example graph. Figure 4.3 illustrates Algorithm 2 with an example.

4.5.3 Double Oracle Graph Contraction

The single oracle methods can prevent us from having to solve the full graph with the complete set of targets. However, it still assumes that we use an exact, exhaustive method to solve the smaller abstracted graphs. For very large problems, this may still be too slow and use too much memory. To address this we introduce the Double Oracle method that also restricts the defender's strategy space when solving the abstracted graphs. This basic idea (a version of column generation) has been widely used in security games literature [75, 113]. Algorithm 3 outlines the procedure.

The outer loop is based on the single oracle method, and gradually adds targets to the restricted set. However, each time we solve the problem for a new contracted graph, we also start from a restricted set of possible paths for the defender. We then solve the "Master" problem (i.e., the original LP), but only with this restricted set of paths. If the solution to this restricted problem already implies that we need to add more targets (because the attacker's payoff is lower than the next best target), we do so and start over with a new, larger contracted graph. Otherwise, we solve a "Slave" problem to find at least one new path to add to the restricted problem, and then go back to solve the Master again. This process terminates when we cannot add any additional paths to the Master that would improve the payoff for the defender (and lower it for the attacker).

To guarantee that we have found the optimal solution, the slave should always return a new path to add that has the minimum reduced cost. The reduced cost of a new joint path J_m is $r_{J_m} = -\sum_i y_i U_a^u(i) P_{J_m,i} - \rho$, where y_i refers to the dual variable of the i^{th} constraint in the original LP (4.3), and ρ is the dual variable of constraint 4.4. The joint path with the most negative reduced cost improves the objective the most. If the reduced cost of the best new joint path is non-negative, then the current solution is optimal. In fact, finding the joint path with the lowest reduced cost is equivalent to solving the following combinatorial



(a) Step 1

(b) Step 2

(c) Step 3

(d) Step 4

Figure 4.3: Example of Single-oracle Algorithm. The numbers shown in the nodes represent the index and the utility of the target. Node 10 is the base node and the defender has only one patrol resource. 4.3a: Original graph (distance limit= 4), which is also the initial current graph G_c. Red lines indicate the greedy route, which determines $\bar{T} = \{10, 9, 8\}$. 4.3b: First restricted graph G_t and the corresponding optimal defender strategy (taking the route $10 \to 8 \to 10$ with probability 0.47), which leads to $u = 4.23$ and $v = 7$. 4.3c: Updated current graph G_c, which is achieved by removing all nodes with utility $\leq u$ (i.e., nodes 2,3,4) and then removing dominated targets (node 7 is dominated by node 9 and node 6 is dominated by node 8). 4.3d: Second restricted graph G_t with updated $\bar{T} = \{10, 9, 8, 5\}$, which leads to u=v=4.58 and the termination of the algorithm.

Algorithm 3 Double Oracle With Abstraction (DO)

Input: original graph G, target utility U_i, $\forall i \in V$

Output: defender mixed strategy x and coverage vector c

1: Sort the targets according to attacker's reward T_{srt}=`sortTargets()`
2: Get the list of initial targets using GCR from T_{srt}, T_{cur} = `GreedyCoverR()`
3: **repeat**
4: Set temporary graph where G_t and all targets $t_i \in G_t$ is also in T_{cur}
5: Generate initial set of paths using GreedyPathR, $s_{cur} = GPR(G_t)$
6: **repeat**
7: Solve SSG for G_t, get mixed strategy x_t, coverage vector c_t, and attacker's expected utility $u = AttEU(G_t, c_t)$
8: Calculate actual attacker's expected utility on original graph $v = AttEU(G, c_t)$
9: **if** $u < v$ **then**
10: Break
11: **end if**
12: Generate paths using $s_t = GreedyPathR()$
13: Append paths $s_{cur} = s_{cur} U s_t$
14: **if** $s_t == 0$ **then**
15: Break
16: **end if**
17: **until** $1 < 0$
18: Find attack target in G $attackTarget(G, c_t)$
19: Add next n e.g. $n = 5$ targets to T_{cur} from $T_{srt} - T_{cur}$
20: **until** $u >= v$ and no more path can be added to s_{cur}

optimization problem:

Definition 4.3 (Coin collection problem)**.** In the coin collection problem, a GSG graph $G = (V, E)$ is given, and each node t_i is associated with a number of coins, denoted as Y_i.

When a node is covered by a patrolling path, the coins on the node will be collected and can be collected at most once. The goal is to find a feasible joint path that collects the most number of coins.

When $Y_i = y_i U_a^u(i)$, the optimal solution of the coin collection problem is the joint path with the lowest reduced cost. The coin collection problem is NP-hard based on a reduction form the hamiltonian cycle problem (details omitted for space). Designing efficient algorithms for finding the optimal or a near-optimal solution of the coin collection problem can potentially improve the scalability of using the double oracle method to find the exact optimal solution to GSG. However, here we are interested in maximizing scalability for the DO approach combined with abstraction, so we designed heuristic methods for the slave that are very fast, but will not necessarily guarantee the optimal solution. More specifically, we use Algorithm 4 as a heuristic approach for solving the coin collection problem.

4.6 Experimental Evaluations

We present a series of experiments to evaluate the computational benefits and solution quality of our solution methods. These experiments will also answer our Research Question 1 by showing that automated removal of lower priority nodes combined with strategy generation, which exploits the sparseness of animal density in the terrain, transforms the game model to a high scalable one. We begin by evaluating the impact of abstraction in isolation, and then provide a comparison of many different variations of our methods on synthetic game instances. Finally, we test our best algorithms on large game instances using real-world data, demonstrating the ability to scale up to real world problems.

4.6.1 Evaluating Graph Abstraction

We begin by isolating the effects of abstraction from the use of strategy generation (using either the single or double-oracle framework). The baseline method solves a graph-based

Algorithm 4 GreedyPathR (GPR)

procedure GREEDYCOVER-COINCOLLECTION
  Initialize best joint path set J_{best}
  for $iter = 0$ to 99 **do**
    if $iter == 0$ **then**
      $Tlist \leftarrow sort(T \setminus B, Y)$ ▷ Get a sorted list with decreasing Y_i
    else
      $Tlist \leftarrow shuffle(T \setminus B)$ ▷ Get a random ordered list
    end if
    $Y_r \leftarrow Y$ ▷ Initialize the coins remained
    for $j = 1$ to R **do**
      Initialize the current patrol route Q_j
      for each target t_i in $Tlist$ with $Y_r(i) > 0$ **do** ▷ Check all uncovered targets
        Insert t_i to Q_j while minimizing the total distance
        if total distance of Q_j exceeds d_{max} **then**
          remove t_i from Q_j
        end if
      end for
      for each target t_i in Q_j **do**
        $Y_r(i) = 0$
      end for
    end for
    if $\{Q_1, ..., Q_R\}$ collects more coins than J_{best} **then**
      update J_{best}
    end if
  end for
  return J_{best}
end procedure

security game directly using the standard optimization formulation, enumerating all joint patrolling paths directly on the full graph. We compare this to first applying our graph abstraction method to the game, and then using the same solver to find the solution to the abstracted graph. We compare the methods on both solution quality and runtime. To measure the amount of error introduced we introduce an error metric denoted by epsilon(ϵ) $= \frac{[U_d(c,a) - U'_d(c,a)]}{U_d(c,a) * 100}$, where $U'_d(c,a)$ is the expected payoff for defender when using contraction and $U_d(c,a) \geq U'_d(c,a)$.

For our experiments we used 100 randomly generated, 2-player security games intended to capture the important features of green security games. Each game has 25 targets (nodes in the graph). Payoffs for the targets are chosen uniformly at random from the range -10 to 10. The rewards for the defender or attacker are positive and the penalties are negative. We set the distance constraint to 6. In the baseline solution, there is no contraction. For different levels of abstraction the number of contracted nodes (#CN) varies between the values: $(0, 2, 5, 8, 10)$. Figure 4.4 shows us how contraction affects contraction time (CT), solution time (ST) and reverse mapping time (RMT). CT only consider the contraction procedure, ST considers the construction of the P matrix and the solution time for the optimization problem, and RMT considers time to generate the P matrix for the original graph from the solution to the abstracted game.

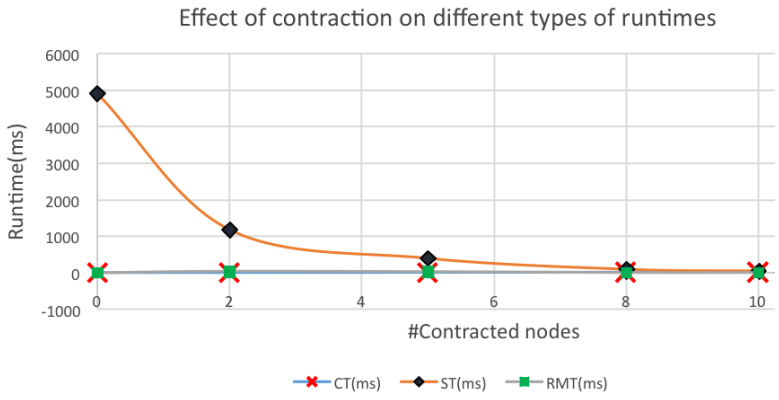


Figure 4.4: Effect of contraction on times CT, ST and RMT

We first note that as the graph becomes more contracted ST takes much less time, as shown in Figure 4.4. The next experimental result presented in Figure 4.5 shows how much error is introduced as we increase the amount of contraction and the amount of time we can save by using contraction.

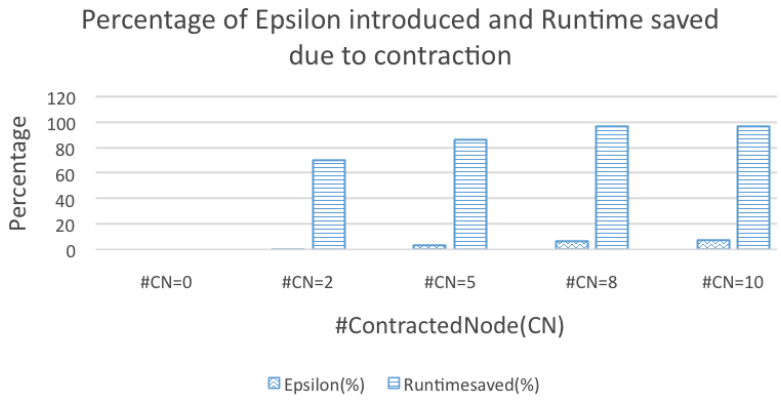


Figure 4.5: Effect of contraction on Epsilon and runtime saved

4.6.2 Comparison of Solution Algorithms

We now present results comparing the solution quality and runtimes of different versions of our solution algorithm on graph-based security games of increasing size. We focus on grid-based graphs, which are typical of open-area patrolling problems like those in wildlife protection domains. For the experiments we generated 20 sample games for each size of game. For simplicity, the distance between every node and it's neighbors is set to 1. The patroller has two resources to conduct patrols in each case, and the distance constraint on the paths varies depending on the game size.

All of the games are zero-sum. We randomly assign payoffs to the targets. In wildlife protection, it is typical for there to be a relatively small number of areas with high densities of animal/poaching activity. To reflect this low density of high-valued targets, we partition the targets into high and low value types, with values uniformly distributed in the ranges of [0, 4] and [8, 10], respectively. We assign 90% of the targets values from the low range,

and 10% values from the high range.

We break down the runtime into three different components: 1) The time to contract graphs, ContractionTime (CT), 2) The time to solve optimization problems, SolvingTime (ST), and 3) the total runtime, TotalTime (TT). All runtimes are given in milliseconds. EPd denotes the expected payoff for defender.

We compare out methods to two baselines that solve the original optimization problem with no contraction by enumerating joint patrolling paths. The first one enumerates all paths and directly solves the problem, while the second algorithm uses column generation to iteratively add joint paths (but does not use contraction). All algorithms that use the path sampling heuristic generate 1000 sample paths. We considered different combinations of the heuristics for both the Single Oracle (SO) and Double Oracle (DO) formulations. In Double Oracle, there are three modules where heuristic approaches can be used: 1) selecting the initial set of targets for the restricted graph; 2) selecting initial paths for solving the restricted graph; 3) in column generation, adding paths that can improve the solution for the restricted graph. The first two modules are also needed in Single Oracle. We discuss the heuristic approaches tested for these three modules. First, for selecting the initial set of targets, we test *GreedyCover1* (GC1) and *GreedyCoverR* (GCR). Second, for selecting initial paths for the restricted graph, we enumerate all the paths (denoted as All paths) for small scale problems. In addition, we test *GreedyPathR* (GPR) and *GreedyPath3* (GP3). When using GPR for selecting initial paths, we use target utility as the number of coins on the targets. *GreedyPath3* (GP3) initialize the set of paths by listing the shortest paths from the base to a target and back to base for each target. Third, to add new paths in column generation, we test GPR and random sampling of paths (denoted as sample path).

We present the runtime and solution quality data for our algorithms as we increase the size of the game, considering game sizes of 25, 50, 100 and 200 targets. Table 4.1 4.2 4.3 and Table 4.4 show the results for each of these four cases, respectively. We had a memory limitation of 16 GB, and many of the algorithms were not able to solve the larger problems within this memory limit. We include only the data for algorithms that successfully solved

Algorithm	#targets left	EPd	CT	ST	TT
DO + GC1 + GPR + LP + GPR	13	6.759	2	11	69
DO + GCR + GPR + LP + GPR	14	5.8845	2	12	65
DO + GC1 + GP3 + LP + GPR	12	7.2095	3	22	44
DO + GCR + GP3 + LP + GPR	10	7.1865	2	15	38
DO + GC1 + GPR + LP + Sample Paths	14	7.481	2	14	165
DO + GCR + GPR + LP + Sample Paths	14	7.3955	2	14	205
DO + GC1 + GP3 + LP + Sample Paths	14	7.605	3	97	267
DO + GCR + GP3 + LP + Sample Paths	14	7.587	2	99	283
SO + GC1 + IC + All paths + LP	12	7.702	1	105	632
SO + GCR + IC + All paths + LP	14	7.702	2	135	827
SO + GCR + IC + GP3 + LP	11	2.05	4	10	33
No contraction + No column generation	25	7.702	0	1417	14140
No contraction + Column generation	25	7.702	0	1480	14661

Table 4.1: Performance comparison, #target=25 and dmax = 8

all of the sample games for a given size within the memory limit.

We note that the baseline algorithms are only able to solve the smallest games within the memory limit. Even for these games, the single and double oracle methods using abstraction are all dramatically faster, and many of the variations come close to finding the optimal solutions. As we scale up the game size, the single oracle methods are not able to solve the game within the memory limit. For the largest games, the double oracle methods without sampled paths are still able to solve the problems to find good solutions, and do so very quickly. The third and fourth variation consistently show the best overall performance, with a good tradeoff between solution quality and speed.

We conduct a second experiment on large, 200-target graphs with the same distance and resource constraints but a different distribution of payoffs. For this experiment we

Algorithm	#targets left	EPd	CT	ST	TT
DO + GC1 + GPR + LP + GPR	30	5.018	9	1313	1981
DO + GCR + GPR + LP + GPR	29	5.8195	7	461	790
DO + GC1 + GP3 + LP + GPR	28	7.4945	14	187	292
DO + GCR + GP3 + LP + GPR	27	7.6415	8	162	261
DO + GC1 + GPR + LP + Sample Paths	30	5.794	9	280	4154
DO + GCR + GPR + LP + Sample Paths	29	6.4185	6	167	3925
DO + GC1 + GP3 + LP + Sample Paths	20	6.8935	6	2194	4499
DO + GCR + GP3 + LP + Sample Paths	23	6.777	6	1570	4330
BA + GCR + IC + GP3 + LP	22	0.75	5	26	1113

Table 4.2: Performance comparison, #target=50 and dmax = 20

Algorithm	#targets left	EPd	CT	ST	TT
DO + GC1 + GPR + LP + GPR	51	6.5135	51	5753	8433
DO + GCR + GPR + LP + GPR	51	6.193	38	2170	3392
DO + GC1 + GP3 + LP + GPR	48	7.0545	52	766	1084
DO + GCR + GP3 + LP + GPR	47	7.2435	37	659	1017
DO + GC1 + GPR + LP + Sample Paths	50	6.098	46	1792	25017
DO + GC1 + GP3 + LP + Sample Paths	20	5.4735	13	2200	4420
DO + GCR + GP3 + LP + Sample Paths	30	5.0745	12	2596	5864

Table 4.3: Performance comparison, #target=100 and dmax = 29

Algorithm	#targets left	EPd	CT	ST	TT
DO + GC1 + GPR + LP + GPR	85	6.5355	345	10360	17904
DO + GCR + GPR + LP + GPR	83	6.501	287	5307	9657
DO + GC1 + GP3 + LP + GPR	72	6.551	270	2658	4156
DO + GCR + GP3 + LP + GPR	70	6.656	189	2274	3603

Table 4.4: Performance comparison, #target=200 and dmax = 45

Algorithm	#remaining targets	EPd	CT	ST	TT
DO + GC1 + GPR + LP + GPR	44	8.621	110	8177	12363
DO + GCR + GPR + LP + GPR	43	8.6085	73	3796	5275
DO + GC1 + GP3 + LP + GPR	40	7.7445	96	595	906
DO + GCR + GP3 + LP + GPR	40	7.7075	70	721	1058

Table 4.5: Performance comparison with 3 partition in payoff, #target=200 and dmax = 45

have three payoff partitions, with the value ranges: $[0, 1], [2, 8], [9, 10]$. The ratio of target values in these ranges is 80%, 10% and 10%, respectively. Table 4.5 shows the results. In comparison with Table 4.5, the DO algorithms (especially variations 3 and 4) are even faster, though in this case variation 1 and 2 do result in higher solution qualities. The distribution of payoffs has a significant effect on the algorithm performance, and as expected, the DO variations with abstraction are most effective when there is a relatively small fraction of important targets and a large number of unimportant ones.

Next we present figures to visualize the runtime differences among different solution algorithms. Again, only algorithms that were able to complete within the memory bound are shown. Figures 4.6a 4.6b and 4.6c show the TotalTime, ContractionTime and SolvingTime comparison respectively among Double Oracle methods and Basic Abstraction Methods with the baseline algorithms. The figures show the same patterns of scalability discussed

previously.

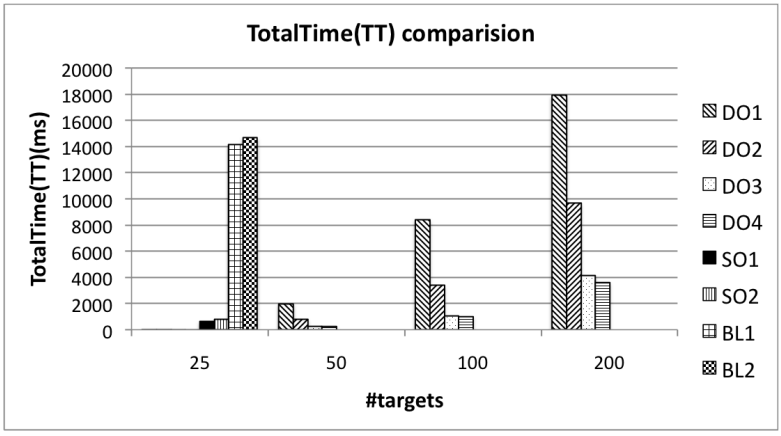
Next, we visualize the solution quality of our proposed algorithms in comparison with the baseline algorithms. The experiment setup is the same as the previous experiment. Figure 4.7 shows that we were able to compare the solution quality properly for $\#target = 25$ since the baseline algorithms were able to finish. The Basic Abstraction methods except the one which uses GP3 were able to compute the exact solution. All of the Double Oracle methods are suboptimal, but typically provide good approximations.

For the next experiments, we only used DO4, which we will mention as DO. We compare the run time between SO, DO and the Baseline techniques in Figure 4.8a. It shows that DO technique takes less time compared to others and for 200 targets SO and Baseline were not able to find a solution because of out of memory exception. Figure 4.8b, 4.8c shows that the DO technique produces expected utility almost as near as other optimal and sub-optimal solvers with a very reasonable amount of time.

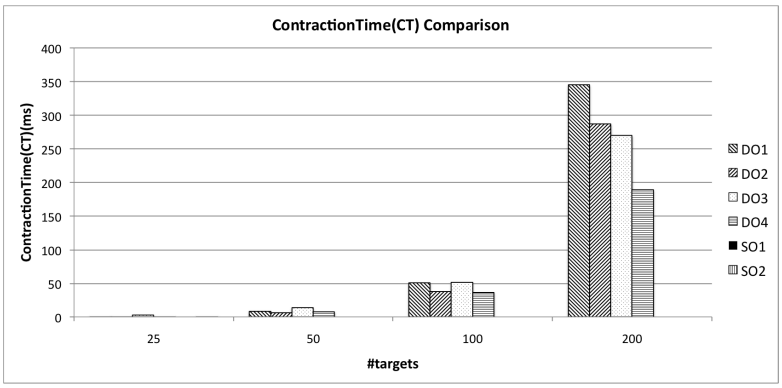
For the final experiments we used real world data. We test our algorithms on grid-based graphs constructed from elevation and animal density information from a conservation area in Southeast Asia. The area is discretized into a grid map with each grid cell of size 50m by 50m. The problem has a large number of targets and feasible patrol routes when considering a practical distance limit constraint (often 5km-20km). We tested with four different game sizes, and the result shows that the proposed algorithm can solve real-world scale GSGs efficiently (see Table 4.6). Only DO4 was used for this experiment since it provides superior performance than others. The payoff range for the targets were $[0, 90]$.

4.7 Conclusion

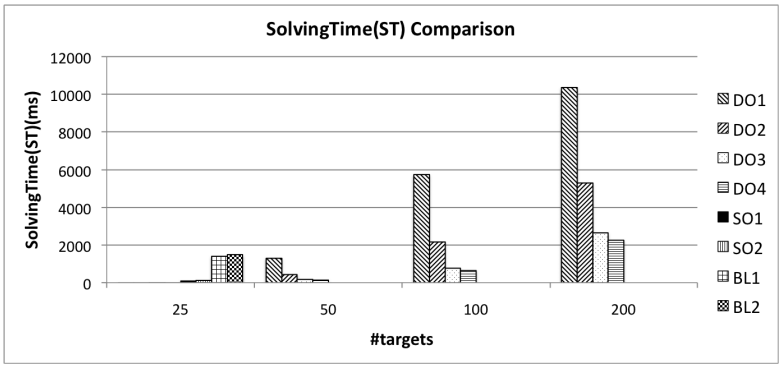
Green security games are being used to help combat environmental crime by improving patrolling strategies. However, the applications of GSG are still limited due to the computational barriers of solving large, complex games based on underlying graphical structures. Existing applications require manual pre-processing to come up with suitably abstract



(a) TotalTime(TT)



(b) ContractionTime(CT)



(c) SolvingTime(ST).

Figure 4.6: Runtime comparison among solvers.

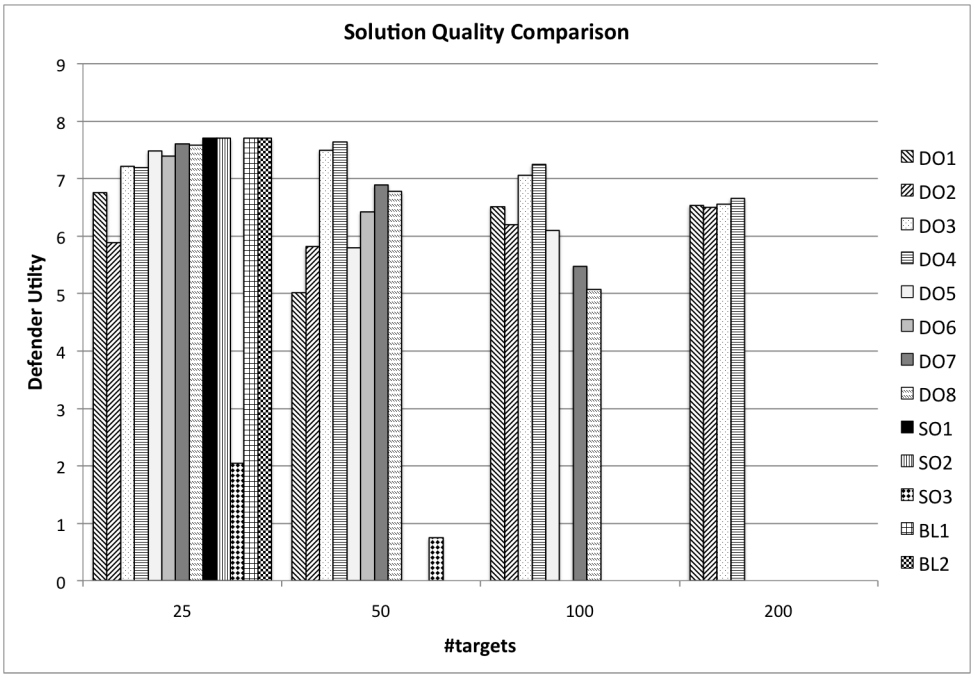
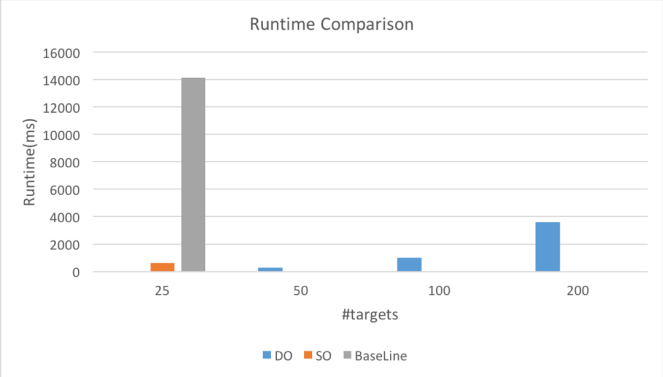


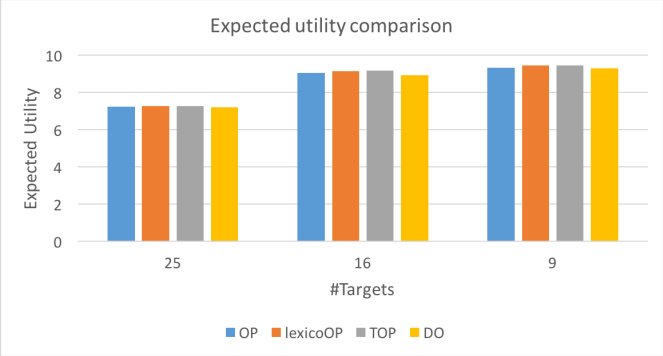
Figure 4.7: Solution quality evaluation

#targets	dmax	#remaining targets	EPd	CT	ST	TT
100	5000	56	25.83	121	303	789
200	8000	88	25.79	67	926	1678
500	15000	92	18.56	1928	1107	4403
1000	18000	100	16.29	12302	2072	18374

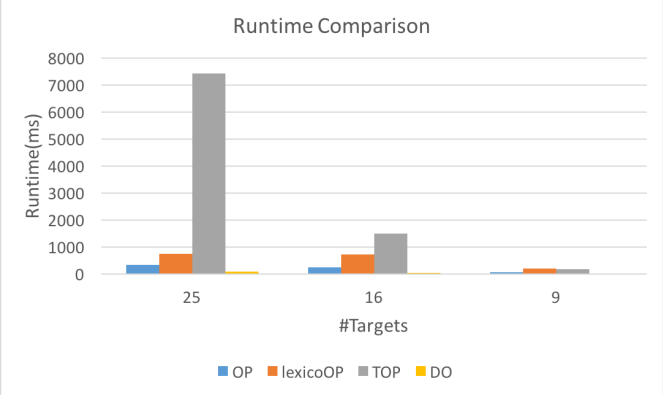
Table 4.6: Results of using abstraction in real world data



(a) Runtime comparison among DO, SO and Baseline



(b) Expected utility comparison between Orienteering Problem (OP), Lexicographic solution for OP with multiple resources (lexicoOP), Team orienteering problem with multiple visitations (TOP) and Double Oracle (DO) solvers



(c) Runtime comparison between Orienteering Problem (OP), Lexicographic solution for OP with multiple resources (lexicoOP), Team orienteering problem with multiple visitations (TOP) and Double Oracle (DO) solvers

Figure 4.8: Runtime and Expected utility comparison among solvers.

games that can be solved by existing solvers. We address this problem by designing the first algorithm for solving graph-based security games that integrates automated abstraction techniques with strategy generation methods. Our algorithm is the first to be able to provide high-quality solutions to very large green security games (thousands of nodes) in seconds by exploiting the characteristic of the grid-based terrain by exploiting the sparseness of animal density. The algorithms will be potentially opening up many new applications of GSG while avoiding the need for some of the arbitrary, manual abstraction stages when generating game models. With additional work to develop fast exact slave algorithms, we should also be able to provide exact solutions using this approach to large GSG. We also plan to investigate approximate slave formulations with performance bounds, using abstraction to compute solution concepts from behavioral game theory such as quantal response equilibrium, and applying our algorithms to real-world applications in green security games.

For my next chapter I move to a different domain, cyber security, where I explore a different research question pertaining to partial independent network structures.

Chapter 5

Subgame Abstraction and Cyberdefense

5.1 Introduction

Most real-world networks are divided into subnets to increase performance and security, but there are limited resources to inspect/harden devices against attacks. Automated Intrusion Detection Systems (IDS) [120] [94] are an essential defense, but it may not be possible to use a costly IDS on every network host [37]. In a network, botnets often spread easily within a subnet using worms that exploit open ports and unpatched vulnerabilities. However, spreading between subnets requires moving through more secure and highly monitored routers that limit connectivity. This locality leads a game model with a particular structure in a Normal Form Game (NFG). We present a game-theoretic model based on this cyberdefense scenario using an NFG for stopping the spread of an attacker (e.g., a botnet) through a network that has a subnet architecture. In this game model, the network administrator acts as the defender and a worm acts as the attacker. The network administrator wants to use his defense mechanism to stop the spread of a botnet by hardening the security in one or more hosts.

While NFG is a very general representation, it is often problematic to solve an NFG for real-world scenarios because enumerating all possible strategies results in an extremely large game. For example, in an enterprise network the large number of hosts and interconnections can lead to intractable NFG models. Solving an NFG is known to be a computationally hard problem [55], and most existing algorithms (e.g., implemented in Gambit [95]) do

not scale well in practice. An increasingly common approach is to apply some form of automated abstraction to simplify the game. The simplified game is then analyzed using an available solver, and the solution is mapped back into the original game. If the reduced game can retain the vital strategic features of the original game, then in principle the solution of the simpler game may be a reasonable approximation of the solution to the original game.

Two works very closely related to an NFG reduction are one by Conitzer et al. [52] and another one by Bard et al. [38]. In the former paper, the authors gave an abstraction technique which can be used in a class of NFGs called *Any Lower Action Gives Identical Utility* (ALAGIU). The authors show that their technique can be applied recursively in ALAGIU games to abstract the game and find approximate Nash equilibrium. Motivated by the approach, we introduce an abstraction technique we call Iterative Subgame Abstraction and Solution Concept (ISASC). To evaluate this technique, we use a class of NFGs where some actions give identical utility that we call *Approximately Identical Outside Subgames* (AIOS). We also introduce a Pure Strategy Nash Equilibrium solution concept called *Minimum Epsilon Bound* (MEB).

Our main contributions are as follows: (1) we model a cyberdefense scenario using NFG that naturally leads to games with AIOS structure, (2) we offer sophisticated non-iterative and iterative algorithms for solving games using abstraction with both exact and noisy AIOS structure, (3) we present experimental evaluation of our algorithms on both generic games and games based on a cyberdefense scenario, showing that our algorithms substantially improve scalability over baseline equilibrium solution algorithms.

5.2 Games with AIOS Structure

A *Normal Form Game* (NFG) is a standard representation in game theory in which the outcomes of all possible combinations of strategies are represented using a payoff matrix. The tuple (N, A, u) represents a finite N-player NFG [114], where each player is indexed

by i. The set of actions (pure strategies) is given by $A = A_1 \times ... \times A_N$, where A_i is the set of actions for player i. Each vector $a = (a_1, ..., a_N) \in A$ is an action profile. An action (kth) for player i is represented by $a_{i,k}$. We extend to mixed strategies $s_i \in S_i$, and use the notation $\pi^i(a_i)$ to refer to the probability of playing action a_i for player i. Each player has a real-valued utility (payoff) function $u = (u_1, ..., u_N)$ where $u_i : A^N \to \mathbb{R}$, extended to mixed strategies as usual by using expected utility.

We will consider abstracted games represented as (simpler) NFGs. For these games, we use the same notation but with a hat to denote that it is an abstracted game, $(\hat{N}, \hat{A}, \hat{u})$. We also use $A_i(O)$ to refer to the set of available actions for player i in NFG O. The set of clusters for player i is denoted using $c_i = \{c_{i,1}, ..., c_{i,m}\}$, where $c_{i,m} \subset A_i$ and $c_{i,m}$ is the m^{th} cluster for player i, and $c_{i,m} = \{a_{i,1}, ..., a_{i,k}\}$. Every action belongs to exactly one cluster, so $c_{i,1} \cap c_{i,2} \cap ... \cap c_{i,k} = \emptyset$.

We now introduce a game structure based on the idea of forming subgames with strong interactions within a subgame, but weak interactions outside of the subgame. We call this *Approximately Identical Outside Subgames* (AIOS), as shown in Figure 5.1. The fundamental idea is to create clusters of strategies for both players that form subgames. Within a subgame, the strategies and payoffs can vary arbitrarily. However, outside of the subgame, the strategies for each player should have payoffs as similar as possible for playing against any opponent strategy, not in the subgame. Games with exact AIOS have identical payoffs outside

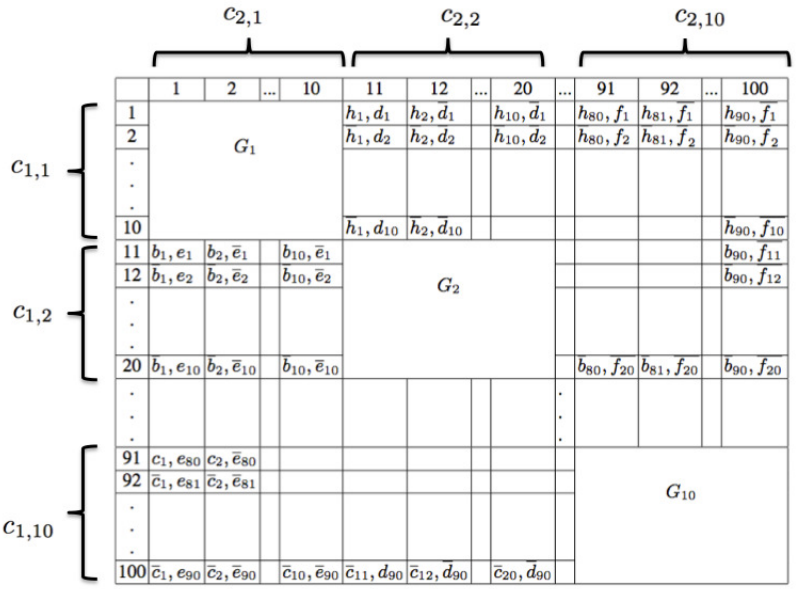


Figure 5.1: AIOS Structure in an NFG

the subgame, while games with noisy AIOS weaken this to allow some variation in the payoffs outside the subgames.

In Figure 5.1, suppose the row player is player 1 and column player is player 2. If player 1 decides to play any strategy from $\{1-10\} \in c_{1,1}$, he needs to worry only about the probabilities assigned by player 2 to strategies $\{1-10\} \in c_{2,1}$. Intuitively this is because if player 2 plays from strategies outside of $c_{2,1}$ the payoff is the same for the row player no matter which action he chooses among $\{1-10\} \in c_{1,1}$. The subgame G_1 is formed by considering only actions in $c_{1,1}$ and $c_{2,1}$.

5.3 A Cyber Defense Game with AIOS

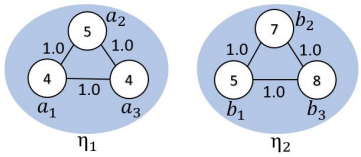


Figure 5.2: Example network with $t_{i,j} = 1$ and $T(\eta_k, \eta_l) = 0$

We now present a cybersecurity scenario for a *botnet* attack where the AIOS (Section 5.2) structure arises naturally. Figure 5.2 shows an example with 2 subnets containing 3 nodes each. A network is a collection of nodes that belong to exactly 1 subnet η_k. Every host has a value v_i. $t_{i,j}$ represents the *intra-transmission probability* for the botnet to propagate from node i to j within the same subnetwork η_k. The *inter-transmission probabilities*, represented by $T(\eta_k, \eta_l)$, is for the botnet propagating from subnet η_k to η_l. The botnet spreads on a new subnet η_l from current subnet η_k and infects the nodes of subnet η_l like a worm maintaining the intra-transmission probabilities. We model a one-shot game where the Defender selects a node i to *defend* (e.g., closing ports, patching vulnerabilities, increasing monitoring). The defend action reduces the transmission probabilities for all edges connected to i and stops any attack that spreads to node i. The attacker selects an initial node to attack, which spreads according to the transmission probabilities (which can

be estimated using simulation).

If the botnet spreads to a defended node and is detected, the Defender pays a cost equal to the total value of the infected nodes (to clean up the attack), but the attacker receives a payoff of zero. If the attack does not interact with a defended node, the attacker receives the sum of the values of all the infected nodes. We estimate the payoff matrix for a particular game using Monte Carlo simulation to estimate the spread of the infection for each pair of strategies.

Attacker

Defender	a_1	a_2	a_3	b_1	b_2	b_3
a_1	-4.0, 0.0	-7.01, 2.27	-6.47, 2.24	-20.0, 20.0	-20.0, 20.0	-20.0, 20.0
a_2	-5.97, 1.97	-5.0, 0.0	-8.0, 2.02	-20.0, 20.0	-20.0, 20.0	-20.0, 20.0
a_3	-9.0, 2.19	-9.0, 2.29	-4.0, 0.0	-20.0, 20.0	-20.0, 20.0	-20.0, 20.0
b_1	-13.0, 13.0	-13.0, 13.0	-13.0, 13.0	-5.0, 0.0	-10.99, 3.75	-11.46, 3.69
b_2	-13.0, 13.0	-13.0, 13.0	-13.0, 13.0	-8.97, 3.18	-7.0, 0.0	-13.0, 3.2
b_3	-13.0, 13.0	-13.0, 13.0	-13.0, 13.0	-12.0, 2.99	-12.0, 3.08	-8.0, 0.0

(a)

Attacker

Defender	a_1	a_2	a_3	b_1	b_2	b_3
a_1	-4.0, 0.0	-7.62, 3.09	-7.12, 3.04	-20.7, 18.58	-20.69, 18.53	-20.7, 18.44
a_2	-6.83, 2.97	-5.0, 0.0	-8.82, 2.93	-20.88, 18.48	-20.84, 18.45	-20.89, 18.44
a_3	-8.6, 2.38	-8.66, 2.33	-4.0, 0.0	-19.89, 18.66	-19.93, 18.77	-19.91, 18.79
b_1	-12.97, 12.17	-12.96, 12.15	-12.96, 12.18	-5.0, 0.0	-10.64, 3.96	-11.28, 4.01
b_2	-14.01, 12.36	-14.02, 12.34	-14.03, 12.38	-9.39, 4.09	-7.0, 0.0	-13.14, 4.06
b_3	-14.41, 12.33	-14.45, 12.26	-14.44, 12.3	-12.39, 3.86	-12.41, 4.08	-8.0, 0.0

(b)

Figure 5.3: (a) Game for Figure 5.2 with $t_{i,j} = 1$ and $T(\eta_k, \eta_l) = 0$. (b) Game for Figure 5.2 with $t_{i,j} = [0.85, 1.0]$ and $T(\eta_k, \eta_l) = 0.10$

Figure 5.3(a) shows the NFG representation for the example in Figure 5.2 assuming $t_{i,j} = 1$ and no edges exist between subnets. In this case, the game has an exact AIOS structure. When the two players play on the same subnet, there is a strategically interesting game. However, when the two players play outside of the same subnets, there is no interaction. Intuitively, this is because the Defender will never be able to detect the Attacker's botnet because no connection exists between subnets. Figure 5.3(b) shows the network seen in Figure 5.2 with a low inter-transmission probability between subnets where $T(\eta_k, \eta_l) = 0.10$ and transmission probabilities within the subnets in the range $t_{i,j} = [0.85, 1.0]$. When we add these weak interactions between subnets (i.e., relatively low transmission probabilities), we have a game with an noisy AIOS structure where actions in different subnets have only limited effects on the payoffs.

5.4 Hierarchical Solution Method

We now describe a solution approach that constructs subgames based on strategy clusters and uses the solutions to these subgames to create a more accurate abstracted game. When games have exact AIOS structure, this will result in finding an exact solution to the original game by composing the results of the subgame solutions. In cases where games have noisy AIOS structure, we propose an iterative solution method that improves solution quality by taking into account error from outside of the subgames.

5.4.1 Subgames

Consider the AIOS example shown in Figure 5.1. Ten subgames correspond to ten pairs of clusters of actions for the players. For example, G_1 is played using clusters $c_{1,1}$ and $c_{2,1}$. Now we consider building an abstracted game by first solving each of the subgames G_1 to G_{10} utilizing any solution concept to get a mixed strategy for each player in each game. The abstracted game will have one action for each player corresponding to each cluster (10 in the example). To fill in the payoffs for each pair of clusters (a 10x10 matrix), we compute the expected payoffs using the mixed strategies for the corresponding clusters (for the subgames, this is the expected payoff from the solution to the game). Figure 5.4 shows the resulting abstracted game R. Next, we solve R using any solution concepts mentioned in section 5.4.3. To get the reverse mapping here we must distribute the probabilities of $c_{1,1}, c_{1,2}, ..., c_{1,10}$ over all the actions in $c_{1,1}, c_{1,2}, ..., c_{1,10}$ for player 1 to get the strategy for the original game (resp. for player 2). Equation 5.1 gives this reverse mapping, where $i \in N, \forall a_{i,k} \in c_{i,m}$. In equation 5.1 the probabilities $\pi^i(a_{i,k})$ on the right-hand side are the mixed strategies for the subgames. We call this approach Subgame Abstraction and Solution Concept (SASC).

	$c_{2,1}$	...	$c_{2,10}$
$c_{1,1}$	$G_{1,1}, G_{1,2}$	...	
.	.	...	.
.	.		.
.	.		.
$c_{1,10}$	...	...	$G_{10,1}, G_{10,2}$

Figure 5.4: Abstracted (hierarchical) Game R

$$\pi^i(a_{i,k}) = \pi^i(c_{i,m}) \times \pi^i(a_{i,k}) \tag{5.1}$$

5.4.2 Noisy AIOS Games

The AIOS structure is strict if we require identical payoffs outside of the subgames. However, it is much more plausible to find approximate forms of this structure. For example, in Section 5.3 we saw how low transmission probabilities between subnets lead to an noisy AIOS game. For a noisy version of AIOS, we define the *delta* (δ) parameter to specify how much variation in the payoffs is allowed outside of the subgames. Let $\delta_{i,k}$ be the maximum payoff difference for any pair of actions in cluster k for player i for any strategy of the of the opponent that is not in the same subgame. δ_i is the maximum of $\delta_{i,k}$ for player i, where k can be from 1 to a number of clusters. Equation 5.2 picks the maximum δ considering all of the clusters and players. Equation 5.3, where $(i,j) \in N, i \neq j$, calculates δ for one cluster $c_{i,k}$ for a player i.

$$\delta = max_{i \in N}(max(\delta_{i,k})), \qquad k = 1, ..., |\hat{A}_i(R)| \tag{5.2}$$

$$\delta_{i,k} = max(u_i(a_{i,m}, a_{j,t}) - u_i(a_{i,n}, a_{j,t})) \tag{5.3}$$

5.4.3 Solving Games

We consider several solution methods for solving games. We consider both pure and mixed-strategy Nash equilibrium, as well as a different concept that directly minimizes the bound on the approximation quality in the original game.

Approximate Pure Strategy Nash Equilibrium

In a Pure-Strategy Nash Equilibrium (PSNE) all players play pure strategies that are mutual best-responses. However, PSNE is not guaranteed to exist. Therefore, we instead

look for the pure-strategy outcome that is the best approximate equilibrium. We first calculate the values of deviations for each action a_i and then select the action profile that minimizes the maximum benefit to deviating.

Mixed Strategy Nash Equilibrium

We also calculate a version of mixed-strategy Nash equilibrium using the software package Gambit [95]. There are several different solvers for finding Nash equilibria in this toolkit. We used one based on Quantal response equilibrium (QRE) [67].

Minimum Epsilon Bounded Equilibrium

When solving an abstracted game, the best analysis may not be finding a Nash Equilibrium, since this may not be an equilibrium of the original game. As an alternative, we introduce *Minimum Epsilon (ε) Bounded* equilibrium (MEB). Instead of considering deviations to clusters of actions (and the average payoff of the cluster), we use the maximum expected payoff for any of the actions in the original game. This heuristic allows for a better estimate of how close the outcome will be to an equilibrium in the original game. The difference in comparison with PSNE is in the calculation of $\varepsilon(a_i^*)$. Equation 5.4 is used to compute the ε for MEB.

$$\varepsilon(\hat{a}_i^*) = max_{\forall \hat{a}_i \in \hat{A}_i, \hat{a}_j \in \hat{A}_j} [\overline{u}_i(\hat{a}_i, \hat{a}_j) - \hat{u}_i(\hat{a}_i^*, \hat{a}_j)] \tag{5.4}$$

In the above equation $\overline{u}_i(\hat{a}_i, \hat{a}_j)$ returns a payoff from an upper bound game $\overline{R}$. Payoffs for the upper-bounded game $\overline{R}$ are computed using Equation 5.5. Equation 5.5 calculates the maximum expected payoff for an abstracted action by reverse mapping to the original actions and calculating the expected payoff for every original action, selecting the maximum one. Next, where $\forall \hat{a}_i \in \hat{A}_i(R), \forall \hat{a}_j \in \hat{A}_j(R), (i,j) \in N, i \neq j$, the equation iterates over all the actions for every player and calculates the payoffs for the upper-bounded game $\overline{R}$. Equation 5.4 cannot be used in the original game because we need an upper-bounded game

where we use reverse mapping. Unless we have an abstracted game, it is not possible to compute an upper-bounded game.

$$\overline{u_i}(\hat{a_i}, \hat{a_j}) = max_{\forall a_{i,k} \in g(\hat{a}_i)} \frac{\sum_{\forall a_{j,l} \in g(\hat{a}_j)} u_i(a_{i,k}, a_{j,l})}{|g(\hat{a}_j)|} \tag{5.5}$$

Double Oracle Algorithm

The Double Oracle (DO) is not a solution concept. It is a technique used to handle massive games. Double Oracle Algorithms [43] [96] utilize the method of column/constraint generation. The idea is to restrict the strategies of all the players and solve the restricted game exactly using the LP [114] for solving an NFG. We used the QRE [67] [95] to solve the restricted general-sum NFG. The QRE gives an approximate Nash Equilibrium.

Counter Factual Regret

Counterfactual Regret Minimization (CFR) [128] is an iterative algorithm to find approximate Nash Equilibrium. In every iteration, it updates the strategies of the players to minimize a weighted sum of regret at each decision. The average strategies then approach NE.

5.4.4 Iterative Solution Algorithm

For games with noisy AIOS structure, simply composing (as above) the solutions of the subgames may not be an equilibrium of the original game. The solution may occasionally play in quadrants of the game that are *not* one of the subgames solved explicitly, which results in an error when the payoffs do not match identically. We now introduce an iterative solution technique that (partially) accounts for this error. After solving the subgames and abstracted game as previously, we now calculate the expected payoff for each strategy outside its subgame. Then, we modify the subgames using this error term added to the payoffs in the subgame and solve them again, and then recalculate the abstracted game

and solve it again. This process results in a sequence of modified solutions that account for the differences in payoffs outside of the subgames from the previous iteration. We call this algorithm the Iterative Subgame Abstraction and Solution Concept (ISASC).

Consider the subgame G_1 in Figure 5.1. We want to internalize the noise outside of the subgame into the payoffs of the subgame. So, before solving G_1, we update the payoffs for both player 1 and player 2. For action $\{1-10\} \in c_{1,1}$, we calculate the expected utility when player 2 does not play the actions in the subgame. That means that when player 1 plays $\{1-10\}$, we calculate the expected utility of $\{1-10\}$, denoted Ω_i, by considering the probabilities of player 2 playing $\{11-100\}$ from the strategy on the previous iteration. Then we update the payoffs of G_1 for player 1 for action $\{1-10\} \in c_{1,1}$ for every $\{1-10\} \in c_{2,1}$ by adding the Ω_i. This process repeats for all strategies in the game.

Pseudocode for updating the subgames is shown in Algorithm 5. Subgames are updated using $[u_i(a_i, a_j) = u_i(a_i, a_j) + \Omega_i(a_i)]$, where $\forall a_i \in A_i(G), \forall a_j \in A_j(G), \forall (i,j) \in N, i \neq j$, line 4-12. Lines 5-7 are used to compute the expected payoff Ω_i, for an action of player i, when player j plays outside of the subgame G. The action set for player i in game G is $A_i(G)$. The probability of action a_j from the mixed strategy for iteration $T-1$ is $\pi_{T-1}(a_j)$.

5.5 Experiments

In the experiment section, we used two criteria to measure the performance of our proposed algorithm: (a) runtime (b) epsilon (ϵ). ϵ measures whether there is an incentive for a player to switch to another pure strategy from the current Nash Equilibrium strategy (which can be either a mixed strategy Nash Equilibrium using QRE or a pure strategy Nash Equilibrium using PSNE, MEB). To compute ϵ of an approximate Nash Equilibrium strategy for player i first we calculate the expected payoff of player i given the approximate Nash Equilibrium strategy of the players. Next, we check whether there is an incentive for player i to switch to a pure strategy from the current approximate Nash Equilibrium strategy (which can be either pure or a mixed strategy Nash Equilibrium). Finally, we

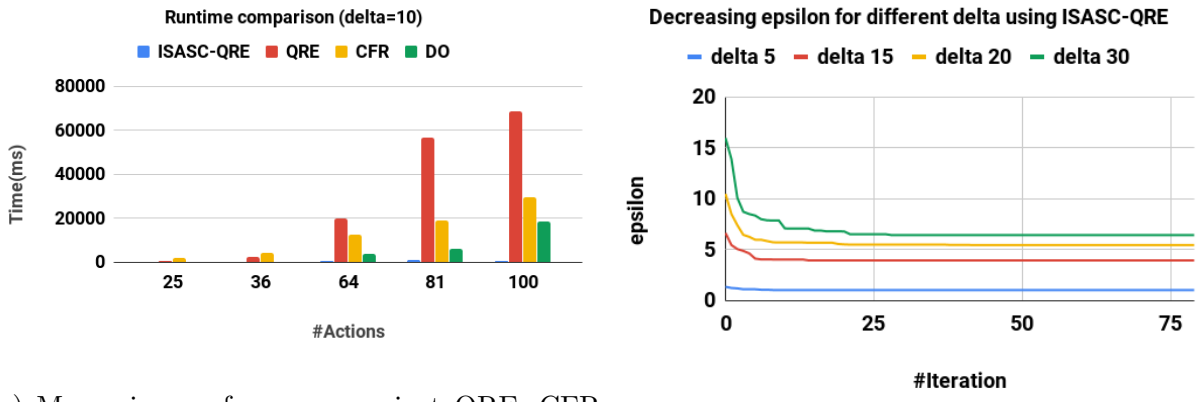
Algorithm 5 Update Subgame Algorithm

Input: Subgame `G'`, original game `G`, player i

1: Subgame actions `o'` = `Actions(G', i)`
2: Opponent actions in `G'`, `p` = `Actions(G, G',` i_{op}`)`
3: Opponent actions outside `G'`, `p'` = `OutActions(G, G',` i_{op}`)`
4: **for** $j \leftarrow 1, o'$ **do** ▷ for every action of player i in `G'`
5: **for** $k \leftarrow 1, p'$ **do** ▷ for every action of opponent $\notin$ `G'`
6: $\omega = \omega +$ `PayOff(j,k,G)` $\times \pi(k)$
7: **end for**
8: **for** $l \leftarrow 1, p$ **do** ▷ for every action of opponent in `G'`
9: Outcome $o =$ `[ j, l ]`
10: `G'(o,i)` = `PayOff(G, o)` $+ \omega$ ▷ update the payoff in `G'`
11: **end for**
12: **end for**

take the maximum of all the players' ϵ which gives us the ϵ for an approximate Nash Equilibrium. In a Nash Equilibrium, there is no incentive to switch to a pure strategy for all the participating players ($\epsilon = 0$). Our next experiments will also evaluate how much scalability we can manage if we take advantage of the AIOS game structure, which corresponds to our research question 1.

For our first experiment we considered 2-player games of different sizes ($\#Actions = 25, 36, 64, 81, 100$) with a fixed $\delta = 10$. For each size, we created 20 different games. For each size, the strategies for each player are partitioned into $5, 6, 8, 9, 10$ clusters with $5, 6, 8, 9, 10$ actions respectively. The subgames are completely random games with payoffs generated uniformly between 0 and 100. The payoffs outside of the subgames are generated randomly with the constraint that in every cluster the maximum payoff difference between the payoffs for the actions is δ for all actions of the opponent that are not part of the subgame (i.e., for every action outside the subgames we add noise).



(a) Measuring performance against QRE, CFR and DO which are applied in the original game

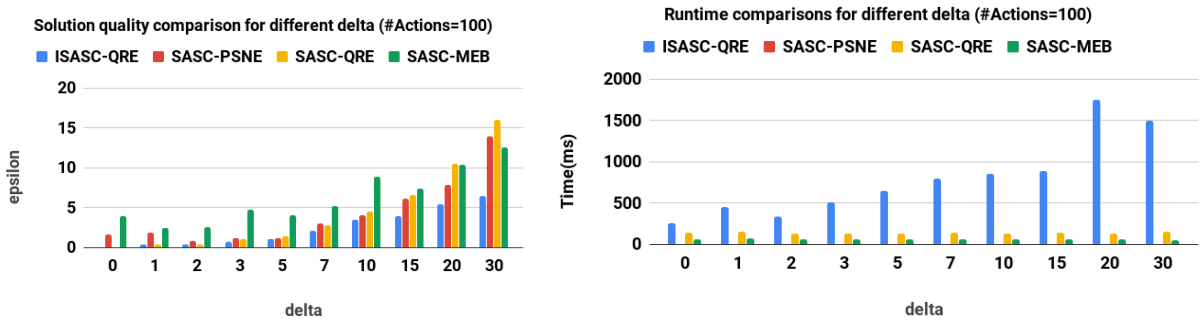
(b) Decreasing ϵ for different δ

Figure 5.5: Measuring performance of ISASC-QRE

We begin by showing that ISASC has benefits when there are limited resources available since ISASC can solve a large game using fewer resources and much more quickly. We compare the runtime performance of ISASC-QRE (ISASC-QRE means we used the ISASC algorithm to solve games where QRE is used to solve the subgames and the hierarchical games) against different solution methods: QRE, CFR and DO, when these different methods are applied to the original game without the use of any abstraction as shown in Figure 5.5a. The results clearly show that ISASC-QRE was able to solve games faster than QRE, CFR and DO by considerable margins.

Our next experiment focuses on showing that the iteration scheme in ISASC-QRE is effective at improving solution quality. For this experiment we created 20 2-player games for each $\delta = \{5, 15, 20, 30\}$ where 100 actions were available for each player. In Figure 5.5b we show the error (quantified by the ϵ in the original game) for different levels of δ as we increase the number of iterations. We can see a clear improvement in solution quality with increasing iterations. We also note that the biggest improvements come in the cases with the largest values of δ.

The next experiment compares the solution quality of iterative (ISASC) and non-iterative (SASC) subgame abstraction techniques: ISASC-QRE, SASC-PSNE, SASC-QRE,



(a) Comparison of solution quality

(b) Comparison of runtime

Figure 5.6: Performance comparisons for ISASC and SASC algorithms

SASC-MEB. For each algorithm, we explicitly mentioned which solution concept is used to solve the subgames and hierarchical games. For example, in SASC-QRE, we used the QRE to solve the subgames and the hierarchical games. The only exception is SASC-MEB, where we used QRE to solve the subgames since MEB can only be used to solve a hierarchical/abstract game. For this experiment, we created 20 2-player games for each $\delta = \{0, 1, 2, 3, 5, 7, 10, 15, 20, 30\}$ where 100 actions were available for each player. The strategies for each player are partitioned into 10 clusters with 10 actions for each: $|c_1| = |c_2| = 10$, $|c_{1,m}| = 10$, $m = 1, 2, ..., 10$. For this experiment, we assumed that the subgames are known to ISASC and SASC algorithms. Figure 5.6a 5.6b shows the results. ISASC-QRE and SASC-QRE does very well in cases with low δ, as expected. However, ISASC-QRE continues to perform better when the values of δ are much more significant. Figure 5.6a 5.6b also show that there is a tradeoff between solution quality and runtime. ISASC-QRE produces the better results but requires more time than SASC-QRE. Overall, the results clearly shows that AIOS games can be highly exploited to achieve a scalable game model using our ISASC algorithm.

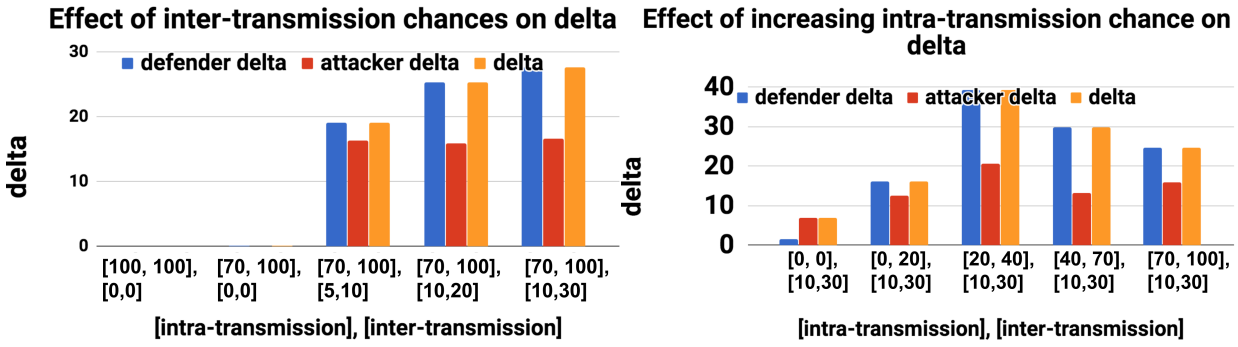
We now consider the more realistic cyber defense games described in Section 5.3. In these experiments we compared our ISASC and SASC algorithms. This section shows that by taking into account the subnet structure, since we know the subnets in advance, our

Parameter	Value range
$t_{i,j} = t_{j,i}$	$[70, 100]$
$T(\eta_i, \eta_j)$	$[10, 30]$
v_i	$[6, 10]$
$c^d(v_i)$	$[1, 3]$
$c^a(v_i)$	$[1, 3]$
$\lvert e_{\eta_k}, e_{\eta_l} \rvert$	1
$\lvert e_{\eta_k} \rvert$	$\lvert e_{min}, e_{max} \rvert$

Table 5.1: Network settings

ISASC and SASC can generate high quality abstracted game model which we can solve using limited computational resources e.g. home computers. For these experiments we did not use any GPU or high performance computational resource. We generated 20 games using the parameter settings shown in Table 5.1. Each parameter is drawn uniformly from the given range. The number of edges in subnet η_l is $|e_{\eta_l}|$ in the range $|e_{min}, e_{max}|$ where e_{min} and e_{max} are the minimum and maximum number of edges respectively. All networks are connected, and the parameters $T(\eta_k, \eta_l)$ and $t_{i,j}$, where $t_{i,j} >> T(\eta_k, \eta_l)$ control worm propagation. We use Monte Carlo simulation for $10,000$ iterations to estimate the payoffs based on the propagation of the attack. Each subnet forms a cluster of actions for our solution methods. Since we already know the subnets for this cyber defense scenario, and thus the subgames, we assumed that the subgames are already known.

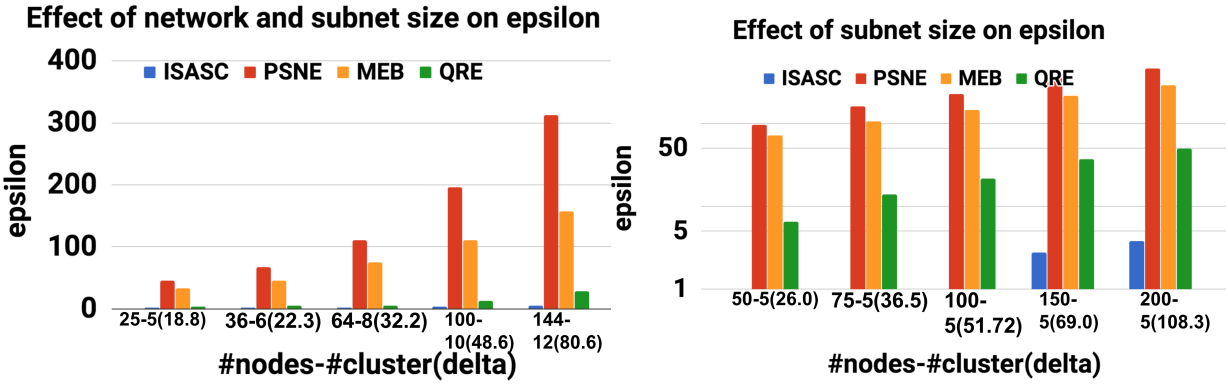
Our first experiment shows how δ varies as we vary the *inter* and *intra-transmission probability* in Figure 5.7a,5.7b. We used games with 50 nodes and 5 subnets with the same number of nodes. We can see in Figure 5.7a that when $t_{i,j} = [100, 100]$ and $T(\eta_k, \eta_l) = 0$ so the subnets are totally disconnected from each other $\delta = 0$. In this case, we can find an exact equilibrium by composing subgame solutions. However, when $T(\eta_k, \eta_l)$ starts to increase δ increases. In both Figure 5.7a and Figure 5.7b, we see that δ reaches a maximum



(a) Impact: $T(\eta_k, \eta_l)$ on δ

(b) Impact: $t_{i,j}$ on δ

Figure 5.7: Effect of transmission parameters on δ



(a) #node and #subnet vs ϵ

(b) Subnet size vs ϵ

Figure 5.8: Performance of ISASC-QRE

when $T(\eta_k, \eta_l) \approx t_{i,j}$ as the spreading of botnet becomes random across the entire network, losing the AIOS structure.

Next, we show how δ and ϵ change when we vary both network size and subnet size. In Figure 5.8a we can see that ISASC-QRE performs favorably compared to the other solution algorithms. Next, we increase subnet size but keep the number of subnets fixed. Figure 5.8b shows that as the subnet size increases δ increases. However, the ISASC algorithm continues to provide better solution quality with very low ϵ for higher δ. In all of the experiments, ISASC gives very high solution quality compared to other algorithms.

5.6 Conclusion

Defending a network against malicious worm requires sophisticated defense mechanism. However, due to large network size and limited resources, it's difficult for a network administrator to harden the security in every host of the network. Solving the game becomes harder due to large action space of the game model. We propose a new class of abstraction methods for NFG based on the AIOS structure. We show that there exist several abstraction-based solution methods that can take advantage of the subnet structure to quickly find solutions to huge games by decomposing them into subgames. For games with only noisy AIOS structure, we show that iterative solution methods can give us very high-quality approximations to the solution of the original game.

Next, I move to the cyber deception domain where honeypots are used to lure the attacker away from real machines and to learn about his behavior and TTP. However, since the attackers are becoming smarter everyday, static honeypot becomes futile to defend the security of a network. In the next chapter I introduce a game theoretic framework where a defender strategically allocate honeypots in a dynamic fashion to achieve his objective.

Chapter 6

Attacker Type Detection

6.1 Introduction

A great deal of effort is devoted to detecting the presence of cyber attacks, so that defenders can respond to protect the network and mitigate the damage of the attack. Going beyond detection, identifying in as much detail as possible what specific type of attacker the defender is facing (e.g., what their goals, capabilities, and tactics are) can lead to even better defensive strategies and may be able to help with eventual attribution of attacks. However, attackers may wish to avoid both detection and identification, blending in or appearing to be a different type of attacker. We present a game-theoretic approach for optimizing defensive deception actions (e.g., honeypots) with the specific goal of identifying specific attackers as early as possible in an attack. We present case studies showing how this approach works, and initial simulation results from a general model that captures this problem. Next, we present a scalable version of the algorithm by reducing the actions space considering domain knowledge and other heuristics. Our initial experiments show that the scalable version performs reasonably well compared to the non-scalable version.

Cyber attackers pose a serious threat to economies, national defense, critical infrastructure, and financial sectors [23, 6, 4]. Early detection and identification of a cyber attacker can help a defender to make better decisions to mitigate the attack. However, identifying the characteristics of an attacker is challenging as they may employ many defense evasion techniques [15]. Adversaries may also mask their operations by leveraging white-listed tools and protocols [118, 122].

There are many existing intrusion detection methods to detect attacks [122, 101]. We

focus here on using honeypot (HP) for detection and identification, though our models could be extended to other types of defensive actions. HP are systems that are designed to attract adversaries [116, 81] and to monitor [101] attacker activity so that the attack can be analyzed (usually manually by experts). There are works on automating the analysis process [107] using host based and network based data for correlation to identify a pattern. Deductive reasoning can be used to draw conclusions about the attacker [108].

Most current approaches focus on detection during an ongoing attack, possibly with some effort to categorize different types of detections. More detailed identification is done during later forensic analysis. Here we focus on formally modeling what types of actions the defender can take to support more detailed attacker identification early in the attack chain, allowing more information to target specific defensive responses. This can be challenging; for example many different attackers may use the same core malware toolkits and common tactics [20, 6, 4, 5]. Attackers may intentionally try to look similar to other actors, and may even change during the course of an attack (e.g., when compromised resources are sold to other groups), leading to a complete change in focus [7, 23, 27].

We focus our model on detecting an attacker type early. An attacker type is defined by an Attack Graph (AG) specific to that attacker that describes his possible actions and goals in planning an attack campaign. An AG represents the set of attack paths that an attacker can take to achieve their goal in the target network [54]. Depending on the observed network and the exploits they have available, the attacker tries to choose the optimal sequence of attack actions to achieve their particular goal. Identification of different attack types in our model corresponds to identifying which of a set of possible attack graphs this particular attacker is using in a particular interaction. The defender chooses deception actions that try to force the attackers into choices that will reveal which type they are early on, which can then inform later defenses. We propose a multi-stage game theoretic model to strategically deploy honeypots to force the attacker to reveal his type early. We present case studies to show how this model works for realistic scenarios, and to demonstrate how attackers can be identified using deception. Finally, we present a general model, a basic solution algorithm,

and simulation results that show that if the defender uses pro-active deception the attacker type can be detected early compared to if he only observes the activity of an attacker. Finally, we use some heuristics considering domain knowledge to reduce the action space to a reasonable amount to make the algorithm more scalable. Our experiments show that using heuristics improves the scalability of the algorithm by reasonable margins.

6.2 Background

AGs represents sequential attacks by an attacker to compromise a network or a particular computer [54]. AGs can be automatically generated using a known vulnerabilities database [73, 103]. Due to resource limitations, the automatically generated AGs are often used to identify high priority vulnerabilities to fix [111, 102]. We use AGs as a library of attack plans that can represent different attackers. The optimal plan for any attacker will depend on his particular options and goals, reflected in the AG. The AG for each attacker will also change depending on the network, including changes made by the defender (e.g., introducing honeypots). We model a multi-stage Stackelberg Security game (SSG)with a leader and a follower. The defender commits to a strategy considering the attacker's strategy. The attacker observes the strategy of the leader and chooses an optimal attack strategy using the AG. The AG of the attackers we considered are defined by initial access and lateral movement actions [16] to reach their corresponding goal node g_i as shown in Figure 6.1. Initial access represents the vectors an attacker uses to gain an initial foothold in the network. We consider the technique called *Exploit Public-Facing Application* [19] for initial access, where an attacker uses tools to exploit the weakness of the public facing systems. Lateral movement is a tactic to achieve greater control over network assets. We consider the example of *Exploitation of Remote*

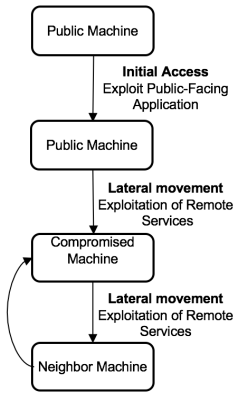


Figure 6.1: A general Attack Graph (AG).

Services technique for lateral movement [22] where an attacker exploits the vulnerabilities of a program. While there may be similarities in the AG for different attackers, having different goals and options available mean that the plans may eventually diverge. The overlap between multiple attack plans is the number of actions that are identical at the start of the plan.

6.3 Case Studies

We present three case studies that consider different types of attackers. We look at different pairs of attackers based on what exploits are shared between them and whether their final objective is the same or not. We use the network shown in Figure 6.2, where a router is R_i, a host is H_i, a firewall is F_i, a switch is S. An exploit $\phi_i(c)$ with cost c on an edge allows an attacker to move laterally if an attacker a_i has exploit $\phi_i(c)$. The cost of using an exploit represents both the time and effort as well as the risk of detection which attackers want to minimize.

An attacker tries to reach a goal by making lateral movements using an attack plan with the minimum cost. If there is more than one minimum cost plan, attackers choose the ones that maximize the overlap with other attackers. We assume that when an attacker reaches his goal the game ends. He also has complete knowledge (e.g. vulnerabilities) about the network, but does not know which nodes are honeypots.

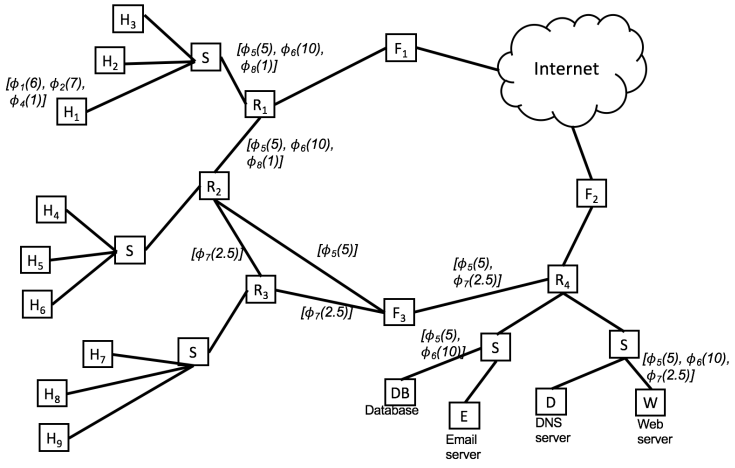
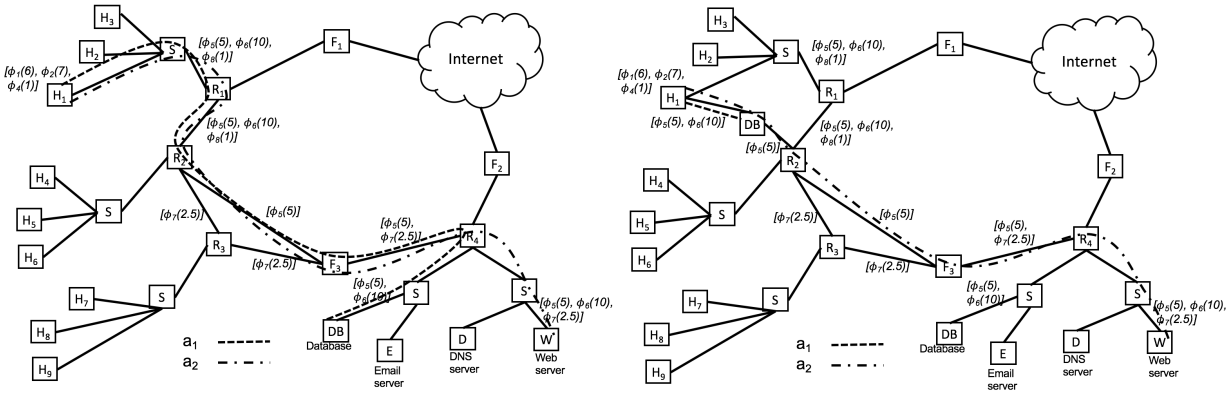


Figure 6.2: Network for case study.



(a) Before deception

(b) After deception

Figure 6.3: Case study 1. Attack plans of the attackers a_1 and a_2 before (a) and after deception (b)

For each case study we first analyze the attack plans based on the AGs. Then we analyze what proactive deceptive action a defender can take to detect the attacker type earlier. We assume that the attacks are fully observable. Since we are interested in scenarios where attackers have common attack plans in their AG, we assume that host H_1 is where all attackers initially enter the network.

6.3.1 Case Study 1: Attackers with Same Exploits but Different Goals

We consider two attackers a_1 and a_2 with the set of exploits $\phi_1, \phi_2, \phi_5, \phi_6$. Goal nodes for the attackers are defined by $g(a_1) = DB$ and $g(a_2) = W$. The attack plans are shown in Figure 6.3a. The defender cannot distinguish which attacker he is facing until the attacker reaches to his goal node. Here, the defender can use a decoy node of either the DB or the W to reduce the overlap in the attack plans by a_1 and a_2. Since both of the attackers have the same set of exploits a defender cannot use any decoy node with vulnerabilities. The attackers have different goals and the defender can take advantage of that situation.

Figure 6.3b shows the use of a decoy DB (with dotted line) between H_1 and R_2 with

unique exploits on the edges to force only the targeted attacker to go through the decoy. Since other attackers will not use that plan, this creates a unique attack plan that can be identified. In Figure 6.3b, we notice that if the acting attacker is a_1, then he will go for the decoy DB. Attacker a_1 does not take the longer path to compromise the DB because he chooses attack plan with minimum cost. To maximize the overlapping attack plan attacker a_2 will choose the plan through the decoy DB instead of R_1 even though the two plans costs the same. In the next case studies, the defender avoids using goal node as decoy to reduce the cost associated with the decoy server.

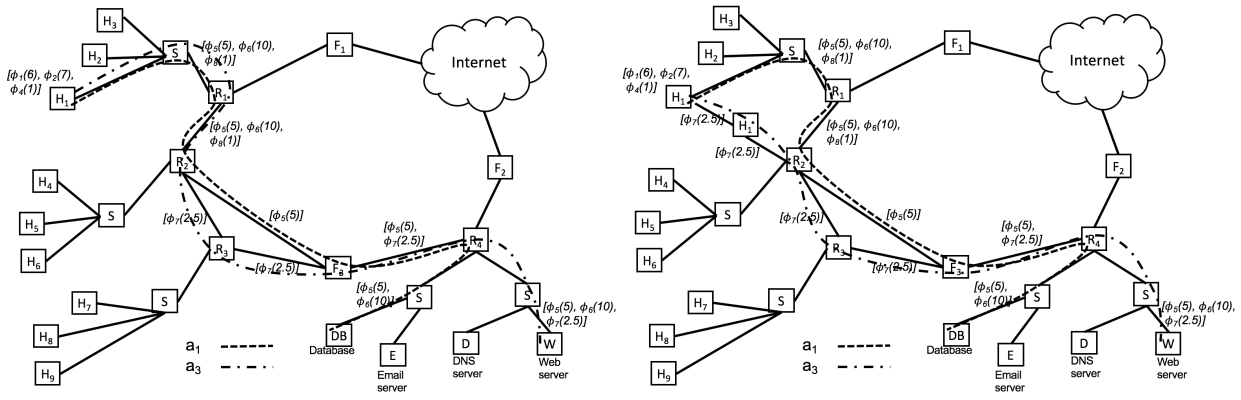
6.3.2 Case Study 2: Attackers With Shared Exploits And Different Goals

Now we consider attacker a_1 with exploits $(\phi_1, \phi_2, \phi_5, \phi_6)$ and $g(a_1) = DB$. Attacker a_3 has exploits $\phi_1, \phi_2, \phi_6, \phi_7$ and $g(a_3) = W$. The attackers a_1 and a_3 compute attack plans as shown in the Figure 6.4a. If the defender just observes, he will be able to detect the acting attacker after R_2. Since the attackers have shared and unique exploits in their possession, these create a non-overlapping path in the middle of their attack sequence, which causes the defender to detect the differentiation between the attackers' attack plans.

However, if the defender uses a honeypot HP as shown in Figure 6.4b the attack sequence of the attackers changes. Attacker a_1 cannot go through the HP due to his lack of exploit $\phi_7(2.5)$ which causes the attackers to diverge at an earlier stage facilitating identification by the defender.

6.3.3 Case Study 3: Attackers with Shared Exploits but Same Goals

Now we consider attacker a_4 with exploits $(\phi_1, \phi_2, \phi_5, \phi_6, \phi_7)$ and $g(a_4) = W$ and for a_5 exploits $(\phi_1, \phi_4, \phi_7, \phi_8)$, $g(a_5) = W$. The attack plans are shown in Figure 6.5a. Attacker a_4 and a_5 executes the exact same attack plan. Even though attacker a_4 could go directly



(a) Before deception (b) After deception

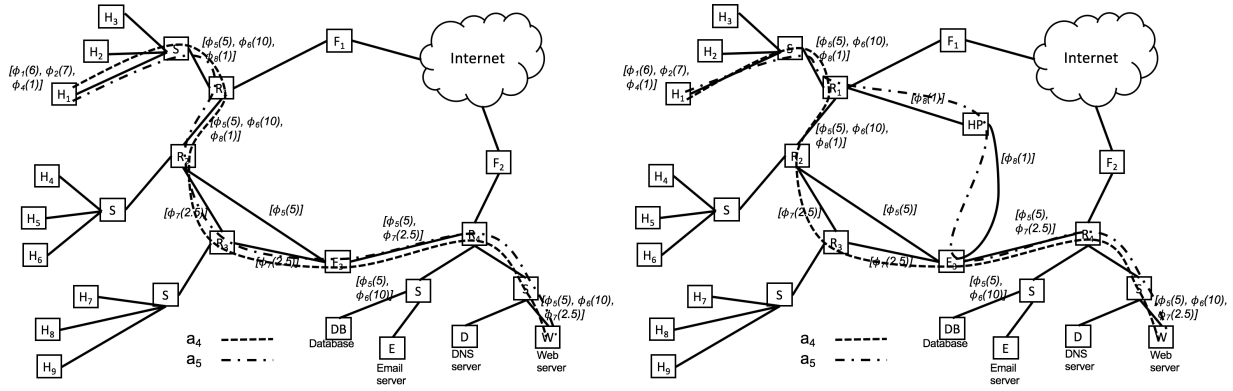
Figure 6.4: Case study 2. Attack plans of the attackers a_1 and a_3 before (a) and after deception (b)

to F_3 from R_2 he chooses the plan to through R_3 minimize the chance of getting type detected. Here, the defender has zero probability to differentiate the attackers.

The defender cannot use any honeypot at the initial stage as in case study 1 or 2 since those will not make any difference in the attackers plan. However, if the defender deploys a honeypot HP as shown in the Figure 6.5b, the attackers choose the plans with the minimum costs which leads to an earlier identification for the defender.

6.4 Game Model

We now present a general game-theoretic framework to model the interactions between the defender and the attacker. There is a network with nodes $t_i \in T$ similar to an enterprise network shown in the Figure 6.2. Each node $t_i \in T$ has a value v_{t_i} and a cost c_{t_i}. Some nodes are public nodes which can be used to access the network. There are goal nodes $g_j \subset T$ (e.g. DB) which are high valued nodes. Each node $t_i \in T$ has some vulnerabilities that can be exploited using $\phi_{t_i} \in \Phi$ on the edges. A node t_i can be compromised from a node t_j using the exploit ϕ_{t_i} if there is an edge from t_j to node t_i and if node t_j allows the use of the exploit ϕ_{t_i} and if an attacker has any of the ϕ_{t_i} exploits. The game has N



(a) Before deception

(b) After deception

Figure 6.5: Case study 3. Attack plans of the attackers a_4 and a_5 before (a) and after deception (b)

attacker types a_i and one defender d. The attackers have complete knowledge about the network and have single deterministic policies $\pi(a_i)$ to reach their goals and some exploits $\phi_{a_i} \in \Phi$. However, he does not know whether a node is a honeypot or not. The defender can deploy deception by allocating honeypot $h \in H$ in the network. The configurations of the honeypots are chosen randomly from real nodes T in the network.

The game starts by choosing an attacker type a_i randomly. In round r, the defender knows the history up to round $r-1$, which is used to update the defender's belief about attacker type. Next, the defender allocates k honeypots to force the attacker to reveal his type earlier. The defense actions also may thwart the plan of the attacker of reaching the goal if the attacker moves laterally to a honeypot or if the attacker does not find any plan to move laterally.

The network change is visible to the attacker in the current round. This can be justified because an APT usually performs reconnaissance again and monitors defender activity after making a lateral movement. The attacker recomputes his attack plan using his AG and chooses an optimal attack plan with the minimum cost from his current position to reach his goal. If there are multiple plans with the same cost, we break the tie in favor of

the attacker where the attackers have the maximum common overlapping length in their attack plans. The game ends if the attacker gets caught by the deception set up by the defender or if he reaches to his goal node.

6.5 Defender Decision Making

In each round of the game the defender updates his beliefs about the attacker types and the attackers' goals. According to Bayes' theorem, given the sequence of lateral movement $seq(t)$ up to node t from the starting node t_p, the probability that the defender is facing attacker a_i is:

$$p(a_i|seq(t)) = \frac{p(seq(t)|a_i)p(a_i)}{\sum_{j=0}^{N} p(seq(t)|a_j)p(a_j)}$$

where $p(a_i)$ is the prior probability of facing the attacker a_i and $p(seq(t)|a_i)$ is the likelihood of the observation $seq(t)$ given that we are facing the attacker a_i. Given the sequence of lateral movement $seq(t)$ up to node t and the attacker type a_i, the probability that the plan of the attacker is $\mathcal{P}_g$ from the start node t_p is:

$$p(\mathcal{P}_g|seq(t), a_i) = \frac{p(seq(t)|\mathcal{P}_g,a_i)p(\mathcal{P}_g,a_i)}{\sum_{\forall g \in G} \quad p(seq(t)|\mathcal{P}_g,a_i)p(\mathcal{P}_g,a_i)}$$

Next, the defender considers all the possible deception deployments $c \in C$ where there are edges $t_m \rightarrow t_n$ from the attacker's last observed position t_{lp} where t_m can be reached from node t_{lp}. Without affecting the existing connections of the network deceptions are deployed between two nodes. The defender has a library of AGs for each of the attackers which he can use to optimize the decision making. We consider three possible objectives the defender can use to make this decision:

Minimizing Maximum Overlapping Length

The defender chooses his deception deployment by minimizing the sum of the attackers' overlapping actions. Another variation would be to minimize attacker's maximum overlap-

ping length with other attackers by considering each of the attackers.

Minimizing Expected Overlapping Length

Minimizing the maximum overlapping length of attack plans may not always focus on all the attackers' attack plans, e.g. if all the attackers have high overlapping (of attack plans) with each other except the acting attacker. Here, the defender can compute the expected overlapping length of the attack plans.

Minimizing Entropy

According to information theory one way to reduce the anonymity between the attacker types is to deploy deception in such a way which will minimize the entropy. If $X_1 = p(a_0)$, $X_2 = p(a_1)$ and $X_3 = p(a_2)$ are three random variables for the attacker types where $X_1 + X_2 + X_3 = 1$, then entropy can be written as follows: $H(X) = -\sum_{i=0}^{i=1} p(a_i) \log_b p(a_i)$ where $p(a_i)$ is the posterior probability for the attacker a_i. For the defender's action the defender chooses the deception deployment which results in the minimum entropy for all the attackers A.

6.6 Attacker Decision Making

Now we present the mixed integer program (MIP) for an attacker, where he chooses the minimum cost plan to reach his goal. Ties are broken by choosing a plan which maximizes the sum of common overlapping length of the attack plans.

$$max \sum_{ij} \quad d_{ij} \tag{6.1}$$

$$0 \leq d_{ij} - \sum_{0}^{me} d_{ijme} \leq 0 \quad \forall i, j \tag{6.2}$$

$$d_{ijme} \leq e_{iem} \; AND \; d_{ijme} \leq e_{jem} \quad \forall i, j, m, e \tag{6.3}$$

$$d_{ijme} \leq d_{ij(m-1)e} \quad \forall i, j, e, m = 1, 2, ..., M \tag{6.4}$$

$$\sum_{em} e_{iem} c_e \leq C_i \quad \forall i \tag{6.5}$$

$$\sum_{m} e_{iem} - \sum_{m} e_{ie'm} = \begin{cases} 1 & \text{if } s \in e \\ -1 & \text{if } g \in e \\ 0 & \text{otherwise} \end{cases} \quad \forall i, \forall t \tag{6.6}$$

$$\sum_{m} e_{ie(m-1)} - \sum_{m} e_{ie'm} = 0 \quad \forall i, \forall t, t \neq s, t \neq g \tag{6.7}$$

Equation 6.1 is the objective function where the attacker computes the maximum sum of overlapping length of attack plans among all the attackers. Constraint 6.2 assigns the sum of the overlapping length between attacker i, j up to move m into d_{ij}. In d_{ijme} e is the edge identifier. Constraint 6.3 computes the overlapping length between the attacker plans where e_{iem} is a binary variable for attacker i representing an edge for edge e (subscript) at mth move. Constraint 6.4 makes sure that the overlapping starts from the beginning of the plans and not in the middle; if two plans start out differently but merges in the middle somewhere. Constraint 6.5 ensures that each attacker chooses a minimum cost plan to reach the goal node. Constraint 6.6 6.7 are path flow constraints for attackers.

6.7 Simulation Results

We want to show that using proactive deception a defender can reveal the attacker type earlier than otherwise. We define an early identification as when the defender is able to use proactive deception to determine the attacker type in an earlier round compared to

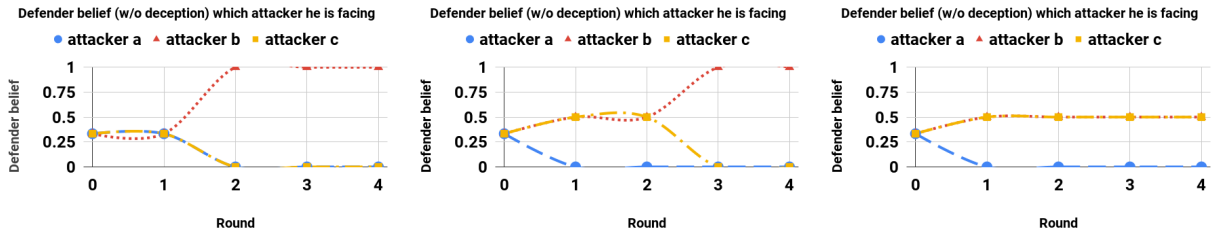
when the defender just observes. Vulnerabilities in the nodes are (indirectly) represented by exploits on the edge between two nodes.

We randomly generated 20 networks, somewhat similar to Figure 6.2, with 18 nodes with values and costs chosen randomly from the range $[0, 10]$ including one public node. We considered exploits $\phi_0, \phi_1, \phi_2, \phi_3, \phi_4, \phi_5$ with cost chosen randomly from the range $[0, 10]$. Next, we assign exploits to the edges in such a way that it allows the attackers to have unique attack plans to their goals except the starting node. Depending on the edge density (number of edges from a node) and shared vulnerability parameters we randomly connect edges with exploits between nodes where the two nodes are in different attack plans of different attackers. We used six honeypots and the vulnerabilities are picked from randomly chosen nodes existing in the network so that the honeypots can act as decoys. The games are limited to five rounds. In each round r the defender d deploys $0 \leq k \leq 2$ decoys. In the attack plan library, we considered three attacker types, a, b, c with different goals.

In the first experiment, we show that depending on different density of edges and shared vulnerabilities between nodes, how early a defender can identify the attacker type he is facing varies. Each of the three attackers, a, b, c has some unique and some shared exploits in their possession. Attacker a has exploits ϕ_0, ϕ_1, ϕ_2. Attacker b has ϕ_2, ϕ_3, ϕ_1. Attacker c has exploits ϕ_4, ϕ_5, ϕ_2. We picked the attacker b as the acting attacker.

Figure 6.6 shows the results. In the first row in Figure 6.6a 6.6b 6.6c defender just observes the attacker actions. As the density and shared vulnerabilities increases it takes more rounds for the defender to identify the attacker type b. In the second row in 6.6e 6.6f the defender deploys honeypots. If we compare the figures of the same edge density and shared vulnerabilities from the two rows it is easy to notice that use of deception facilitates early identification except Figure 6.6d where it was the same. However, an increase in edge density and shared vulnerabilities between nodes harms performance.

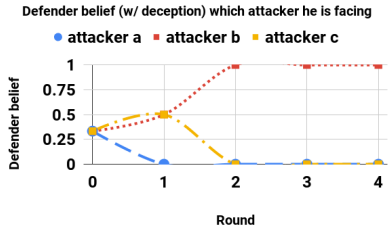
The second experiment is the same as the first except that we kept the edge density and shared vulnerabilities between nodes fixed to 40% and we varied the shared exploits between the attackers. We chose the attacker c as the acting attacker. We can observe in



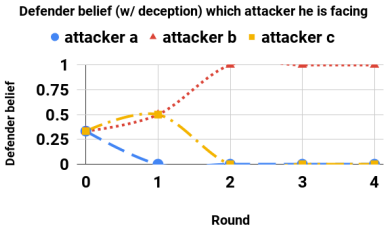
(a) Edge density and shared vulnerabilities 20%

(b) Edge density and shared vulnerabilities 40%

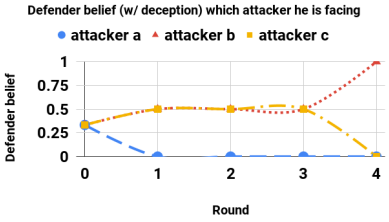
(c) Edge density and shared vulnerabilities 80%



(d) Edge density and shared vulnerabilities 20%

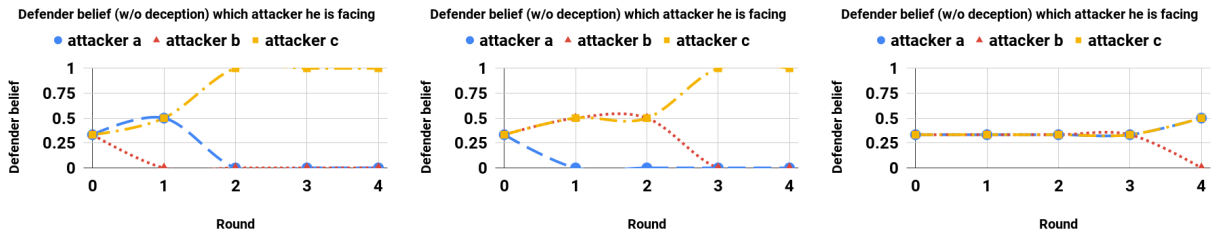


(e) Edge density and shared vulnerabilities 40%



(f) Edge density and shared vulnerabilities 80%

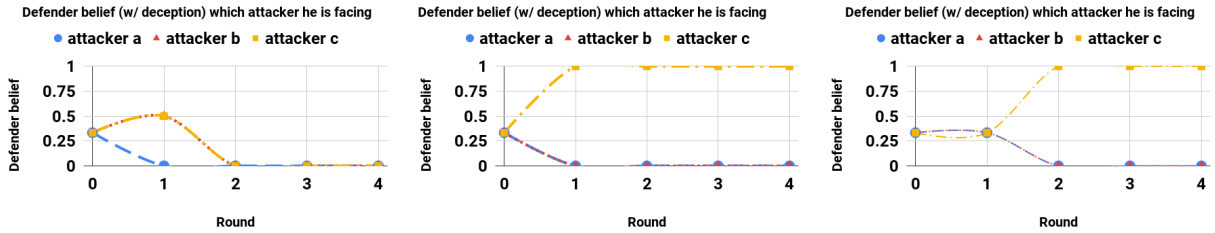
Figure 6.6: Comparison between just observation (first row) and use of pro-active deception (second row). The shared exploits are fixed to 40% between attackers.



(a) Unique exploits for the attackers

(b) 40% shared exploits between attackers

(c) All attackers have the same set of exploits

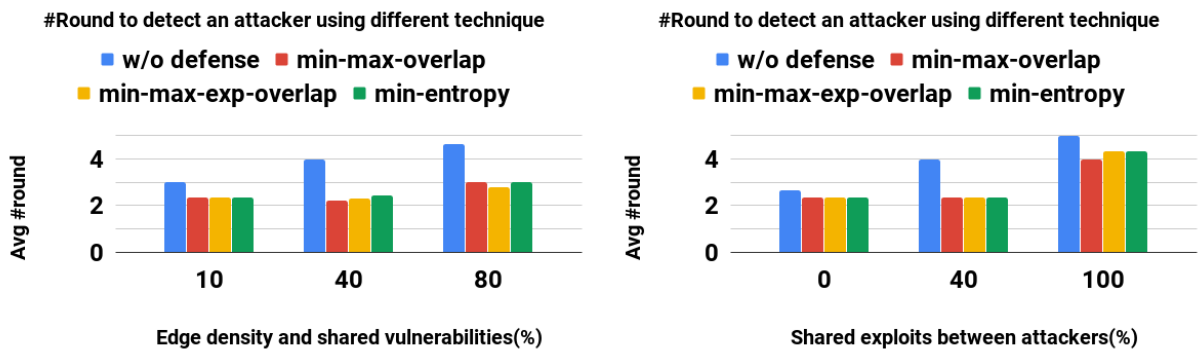


(d) Unique exploits for the attackers

(e) 40% shared exploits between attackers

(f) All attackers have the same set of exploits

Figure 6.7: Comparison between just observation (first row) and use of pro-active deception (second row) We increase shared exploits between the attackers. The edge density and shared vulnerabilities between nodes are fixed to 40%.



(a) (b)

Figure 6.8: Comparison between different techniques used by the defender.

Figure 6.7a 6.7b 6.7c that as we increase the sharing of exploits between the attackers it takes longer for the defender to identify the attacker type c. When all the attackers have the same exploits the defender was unable to identify the attacker type even at around 4 without using any deception. However, in the second row in Figure 6.7d 6.7e 6.7f as the defender strategically uses deception, identification of attacker c happens earlier. The performance of the early identification decreases as the shared exploits between the attacker's increases. Another observation is noticeable in Figure 6.7d: the defender was not able to identify the attacker type, however, the attacker did not find any policy to continue its attack. This shows that the use of strategical deception can also act as a deterrent for the attackers.

For our last experiment, we compared different techniques a defender can use to facilitate early identification; minimizing maximum overlap: *min-max-overlap*, minimizing maximum expected overlap: *min-max-exp-overlap* and minimizing entropy: *min-entropy*. Data are averaged for all the attacker types we considered. Figure 6.8a 6.8b show on average how many rounds it took for the defender to identify the attackers using different techniques. In both figures, we can see that using deception facilitates earlier identification. We did not notice any particular difference between different techniques except when all the attackers have the same exploits, and in that case *min-max-overlap* performed better. From all the experiments it is clear that use of deception will speed the identification of an attacker

which is very important in cybersecurity scenarios as different real-world attackers e.g. APTs can lay low for a long time and detecting the attacker early can facilitate informed decision making for the defender.

6.8 Scalability

In our current game model, the defender makes a move after considering the attacker's last known position and every possible way k honeypots can be allocated between two real nodes in the network. Let's define $\sigma(i, j)$ as a slot where a honeypot can be allocated between node i and node j. The current algorithm makes sure that node i is always one hop distance away from the attacker's last known position and node j does not include the goal nodes to exclude trivial cases.

Using this approach the number of slots of $\sigma(i, j)$ where k honeypots can be allocated increases very quickly with both network size (branching factors) and k which makes the algorithm not very scalable. For our initial experiments, we used a network of size 18. Now we will present a simple heuristics that allows the algorithm to handle larger instances of networks compared to our initial approach.

6.8.1 Heuristics

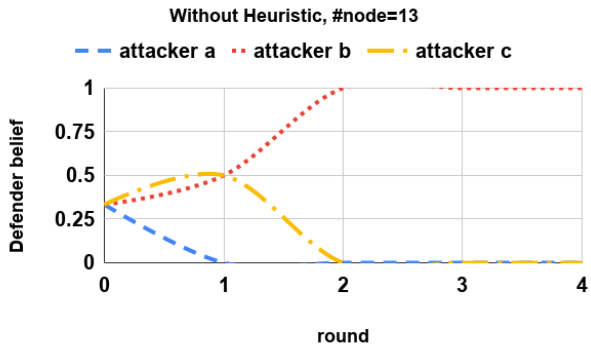
In the real world, sensors, intrusion detection systems and forensic analysis are used to collect data and to analyze the alerts and to understand the TTP used by an attacker. Utilizing the same tools and analytics it is also possible to form a belief on the approximate time interval of the attacks of an attacker and his preferences towards different features of the network, for example, OS, application, ports, hardware etc. Rather than just focusing on TTP used by an attacker the defender can try to understand the attacker behavior which drives the attacks and the attacker's propagation throughout the network. If the defender knows the last position of the attacker in the network, the time interval and preferences and attacker behavior can be used to get an estimate on his future attacks in the network to

form a radius from the last known position of the attacker to consider the slots for honeypot allocation. Using these estimations on the attacker's future positions in the network many of the unnecessary *slots* also can be filtered out.

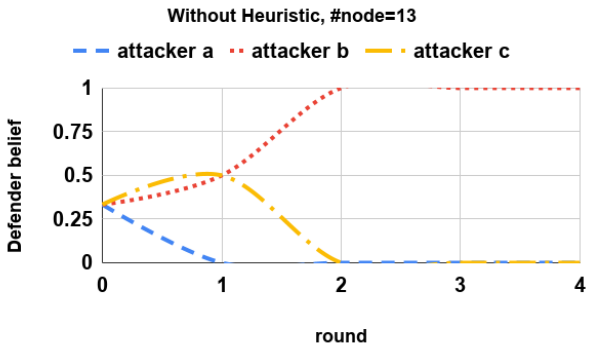
In our game model, we do not capture this complex behavior of attacker preferences, behaviors, and interval of attacks. We simplify it by assuming that the defender knows the attacker's last position in each round and defender also knows that the attacker only moves 1 hop per round. Using these assumptions the defender can consider *slots* $\sigma(i, j)$ where the distance between node i and node j is always 2 hops and node i is always 1 hop away from the last known position of the attacker. It is also unnecessary to consider *slots* $\sigma(i, j)$ where node i and node j are only 1 hop away since it will only introduce costs to the paths and we assume that the attacker chooses a path which minimizes his costs. More unnecessary slots can be eliminated from considerations utilizing domain knowledge; for example, the defender knows that the attacker only moves forward (which is not realistic) however, this is one of our assumptions of the game model; as a result it is not necessary to consider *slots* which are behind the attacker's last known position.

6.9 Evaluation of Heuristics

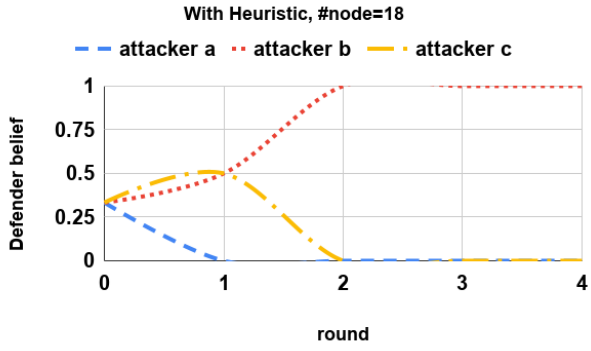
Our goal is to evaluate the heuristics we used to reduce the number of possible ways honeypots can be allocated in the network to make the algorithm more scalable compared to our previous experiments. In the first experiment, we evaluate the solution quality of the algorithm with and without heuristics by comparing how early the defender can identify an attacker. We used different sizes of networks with different limits on the number of rounds the players can play. Each of the three attackers we considered, a, b, c, has some unique and some shared exploits in their possession. Attacker a has exploits ϕ_0, ϕ_1, ϕ_2. Attacker b has ϕ_2, ϕ_3, ϕ_1. Attacker c has exploits ϕ_4, ϕ_5, ϕ_2. We picked the attacker b as the acting attacker. We kept the edge density and shared vulnerabilities between nodes to 40%. Results are averaged over 20 game instances. In the Figure 6.9, the first row, Figure 6.9a,6.9b computes



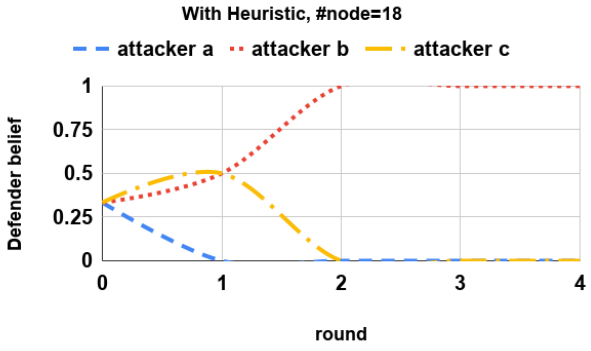
(a) Solution quality without heuristics #nodes= 13



(b) Solution quality without heuristics #nodes= 18



(c) Solution quality with heuristics #nodes= 13



(d) Solution quality with heuristics #nodes= 18

Figure 6.9: Comparison between algorithms without heuristics (first row) and with heuristics (second row). Edge density and shared vulnerabilities between nodes 40%

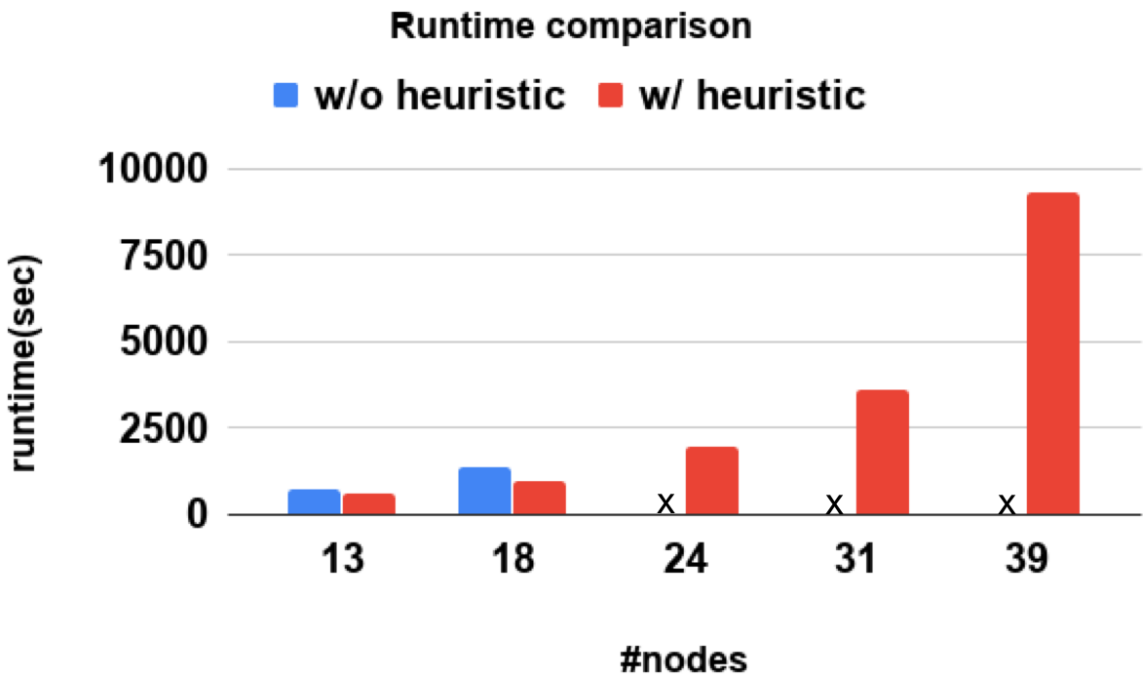


Figure 6.10: Comparison between run times of the algorithms for different sizes of networks. Edge density and shared vulnerabilities between nodes 40%

the solution without using the heuristics and the second row, Figure 6.9c, 6.9d computes the solution using the heuristics to reduce the action space. As we can see, the solution quality did not degrade for the experiments we conducted. However, currently, we have no proof to guarantee an optimal solution with the use of heuristics.

In the next experiment, we compare the run time between the algorithms using without and with heuristics for different sizes of networks and the same setup as the previous experiment. Run times are averaged over all the attackers for a particular size of games. As shown in Figure 6.10, for smaller instances the algorithm with heuristics can compute the solution much quicker than if we do not use any heuristics. For larger instances of games if we do not heuristics the algorithm runs out of memory very quickly (as shown by 'x') whereas using the heuristic we can compute the solutions.

The experiments above clearly show that if we reduce the action space, it is possible to make the algorithms faster. That answers the research question 1: by revealing more information to the defender it is possible to reduce action space for the defender. However, there can be other ways to increase the scalability performance of the algorithms. For example, we can reduce the graph itself to construct an abstracted version of the game. However, we will consider that as our future work.

6.10 Conclusions

Detecting and identifying attackers is one of the central problems in cybersecurity, and defensive deception methods such as honeypots have a key role to play, especially against sophisticated adversaries. Identification is an even harder problem in many ways than detection, especially when many attackers use similar tools and tactics in the early stages of attacks. However, any information that can help to narrow down the goals and likely tactics of an attacker can also be of immense value to the defender, especially if it is available early on.

We present several case studies and a formal game model showing how we can use

deception techniques to identify different types of attackers represented by the different attack graphs they use in planning optimal attacks based on their individual goals and capabilities. We show that strategically using deception can facilitate significantly earlier identification by leading attackers to take different actions early in the attack that can be observed by the defender. Our simulation results show this in a more general setting. In future work we plan to explore more specifically how this type of information can be used to respond dynamically to specific attackers during the later stages of an attack. We also plan to investigate how this model can be extended to different types of deception strategies, integration with other IDS techniques, as well as larger and more diverse sets of possible attacker types.

Current test beds do not support the capabilities offered by game theoretic or ML algorithms for dynamic deception. Moreover, there are current test beds also do not support the evaluation of cyber deception algorithms. So, for my next chapter I present a testbed that supports different capabilities offered by the game theoretic framework of dynamic cyber deception where we can evaluate the effectiveness of a dynamic cyber deception algorithm.

Chapter 7

Evaluating Game Theory in Realistic Cybersecurity Scenarios

7.1 Introduction

While scalable solution methods allow us to apply game theory to realistic problems, we also need to validate that the models can capture realistic problem constraints and considerations, and produce useful results in real-world problems. This goes beyond evaluating the algorithms themselves into considering the applicability of the models, evaluating their assumptions, and testing them in more realistic settings.

Cyber deception using honeypots provide a unique way to maintain network security. It lures an attacker from the real machines and gives a unique opportunity to look into the techniques tactics and procedures used by an attacker. This opportunity (through new intelligence) allows for improved IDS and IPS. However, previous approaches of utilizing honeypots are static upon receiving any alerts, indicators, or any artifacts and does not take into consideration that an attacker can observe and learn to predict the honeypots. Recently, dynamic deception strategies using Game Theory and Machine Learning have been developed [39, 68] to defend against intelligent attackers which provides less predictable honeypots, targeted honeypot allocation strategies, ability to manipulate an attacker and improvement of honeypots.

However, applying these intelligent algorithms for dynamic deception strategies faces unique challenges when we want to apply them in realistic settings e.g. the physical constraints of observation (monitoring) and dynamic allocation of honeypots. More specifically,

effective monitoring of network traffic and hosts, delays introduced by honeypots, the effectiveness of deception, and many more. So, we must evaluate the Game Theoretic and Machine Learning algorithms in a testbed that allows the action space required by the algorithms and addresses the gap between the theory and practice. Moreover, the testbed itself needs to be scalable, repeatable, accessible.

In this final chapter of my dissertation, I present an initial attempt to apply the game theoretic model of the attacker identification project 6 into a realistic scenario using the Cybersecurity Deception and Experimentation System (CDES)[1]. The CDES testbed is an extension of the Common Open Research Emulator that is an already established network emulator. CDES is scalable, follows a modular design, and provides the opportunity to integrate algorithms and other software that can be utilized for proper evaluation. Most importantly it supports the capability of dynamic deception strategies required by the Game Theory and Machine Learning algorithms. The experiment section shows that a defender can use game theory to defend his network by identifying an attacker identity by utilizing dynamic deception.

APT poses a serious threat to national security, infrastructure, society, economy, academia, and even political campaigns [1]. APTs are usually sponsored by a nation-state to achieve a mission goal. They are highly skilled with sophisticated capabilities [29] [17]. For example, before launching an initial attack to the target they gather technical and people information on the target organization to find weaknesses so that they can develop capabilities to infiltrate the system. Next, using the developed capabilities they launch the initial attack to have a foothold in the target network. For initial access, an APT can also launch a Supply Chain Attack [23, 7] that removes the necessity of phishing or drive-by attacks [25]. Once an APT has a foothold inside a network it usually employs sophisticated TTP until the mission goal is obtained. For example, an APT can use various defense evasive techniques to fool the defense mechanisms [26]. They can learn by monitoring the network and collecting data from sensors and then modifying their executable using Artifact kit [9]. They

[1]Submitted for review

can also learn the environment so that they build their own VMs where they test/emulate their new attack capabilities to find the most effective strategy for the next attack [9].

An APT can persist without any detection in the target network for an extended period of time. A defender must detect and identify an APT earlier in its life cycle since the defender can make a better-informed decision. However, it is not straight forward since many APT can employ similar TTP and they can buy the same tool-kits from the dark web. As a result, even after detecting an APT the problem of identifying and stopping an APT still could remain unsolved [39]. An APT can also impersonate [6, 5] or deploys artifacts pro-actively to inhibit the forensic investigation. They may employ many defense evasion techniques [15]. Adversaries may also mask their operations by leveraging white-listed tools and protocols [118, 122]. Current approaches rely on detection during an attack and detailed identification is done in a post forensic analysis. The industry also lacks the use of pro-active measures to detect and identify an attacker earlier. Usually, most of the process to observe and match behavior pattern to detect attacker activity can be difficult due to the aforementioned reasons.

I present an operational game-theoretic model where the defender actions can facilitate detailed attacker identification early in the attack cycle. The model we present is based on the game theoretic model introduced in the work by Basak et al. [39]. When describing the defender algorithm we also show that an effective NIDS, HIDS can reduce the action space of the defender to make the game model scalable. We first give a summary of the existing game-theoretic model, then we present the operational game model. Then we do emulation experiments in the CDES testbed to show that the game theory can is used by a defender to detect and identify an attacker earlier in the attack chain through dynamic deception and manipulation of the attackers.

7.2 CDES Testbed

The target sector of an APT can vary depending on its current mission goals [23, 7]. As a result, a single type of network is not enough. If we want to test different capabilities of a network against these intelligent attackers, we need a network that can support dynamic metamorphism into a necessary topology at that moment. The existing testbeds are manually constructed depending on the need for the validation procedure [50]. It's time-consuming. Also, most of them are commercial so it is not open to the cybersecurity community where lack of data is very prominent.

We tackle this issue by using the CDES testbed which can facilitate experiments to operationally verify the defense capability of a heterogeneous network regardless of its domain (e.g. military, ad-hoc, wireless, enterprise, university, etc.) that is autonomous, self-sustained, resilient and pro-active. It is built upon already established CORE which provides a real network environment. CDES allows its users repeatable experiments, integration of algorithms and software, and their modifications. It has a GUI for building realistic network scenarios and interfaces for easy modification of different components of its modules.

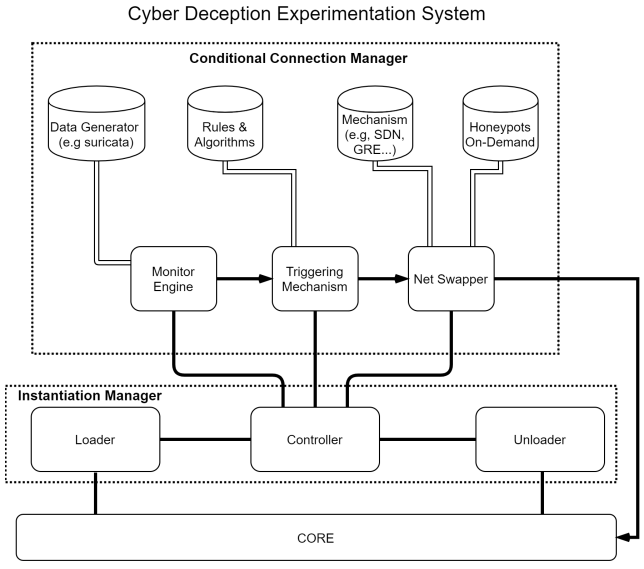


Figure 7.1: CDES Interfaces and Data Flow

CDES has three main modules as shown in the Figure 7.1: CORE, instantiation manager, and conditional connection manager. The CORE provides the underlying emulation environment. The instantiation manager loads a scenario and initiates every parameter of that scenario provided by a user. This manager allows for the initiation of the controller which has a monitor, a trigger, and a network swapper component which are part of the conditional network manager. The monitor module collects traffic data and raises alerts based on the NIDS and its rules. The trigger module keeps track of the alerts raised and can make decisions based on the alerts. This is the module that can be used to integrate artificially intelligent algorithms for dynamic deception strategies. The network swapper module receives commands through API to dynamically change the honeypots and activate or deactivates honeypots by sending messages to the CORE.

Besides early detection, and identification of APT using game theory this autonomous dynamic property using game theory can open doors for many other possibilities. For example, we can collect and publish data based on different emulations in CDES that can be used by the community. We can also support the capability of reducing a load of computation because of the vast amount of data by strategically selecting different parts of a network to sense, collect, and compute using game-theoretic algorithms.

7.3 Moving to an Operational Game Theoretic Model

We now describe the factors we considered to transform a theoretic framework into an operational one because in a theoretic model any aspect of a real-world computer network is abstracted to make the model less cumbersome. However, it is not possible to make a theoretic game model as complex as a real one. We discuss each of the components of the game theoretic model and what we need to consider to transform it into an operational game theoretic model.

The networks in the game theoretic model consist of nodes and edges. The topology of the network is sufficient to represent many aspects of a real-world network. However, a

real-world enterprise network is usually segmented into subnets, DMZs, internal networks, external networks, and there can be firewalls and IDS, IPS, etc. which were not considered in the previous work. The firewalls make it possible to segment the network into DMZ; that means they control the network packet flow by blocking and allowing traffic flows from different directions. In our operational game model, we consider these aspects of a real-world computer network. We also consider IDS that acts as the situational awareness module for the defender.

The game theoretic model has a set of attackers and one defender. Each of the attackers has a set of exploits that they can use to move laterally from one node to another. The attacker makes an attack plan (from the current node to the goal node) based on his AG assuming that he knows the current topology of the network. However, in reality, different firewalls will prevent an attacker from doing so unless the attacker has already compromised all the nodes of the network or has done reconnaissance previously.

In the operational model, we break down the action of an attacker into more realistic steps. First, the attackers need to scan the surrounding hosts and open ports. Then they need to scan for vulnerable applications running on the open ports. We can think of these scanning steps as the reconnaissance phase. After gathering necessary information the attacker decides on which vulnerability to exploit on which host based on his preference and objective. We also limited the attacker's observation of the network based on firewalls. So, we cannot assume that the attacker has complete knowledge about the topology of the network. Since it is not realistic to construct an attack plan to the goal node from the current node the attacker uses some other measurement to make his decision.

We used two metrics to evaluate available options for decision making of an attacker: 1) preference for using different exploits 2) level of alerts. That means an attacker uses his preference for exploitation to move laterally. If there is a tie, he breaks the tie using the inclination of raising minimum alerts. Instead of making an attack plan from the current node to the goal node, an attacker uses his local observation and evaluation metrics to decide for the current stage. We describe more about the attacker action space when we

formally define the attacker for the operational model.

The theoretic model assumes that the defender can observe the attacker's action until the last round. Based on history he updates his belief and then deploys honeypot. In the real world that does not happen. In the real world, the defender usually uses sensors in the hosts and NIDS, HIDS to keep himself aware of the state of the network. However, this is a hard task since there is legitimate traffic in the network and the inherent characteristics of the network protocols prevent the defender from detecting the attacker activity easily. For our operational model, we write rules to detect network traffic patterns which alert on attacker scanning and exploitation activities. More about the defender's observation (detection) stage will be described in the next section.

Next, the defender deploys honeypots. To evaluate each honeypot configuration and deployment settings the defender utilizes the attacker models. We also assume in the operational model that the defender is aware of all the attacker models. This allows the defender to measure the effectiveness of different honeypot deployments. In the game theoretic model, the defender minimizes the overlapping length of the attackers' attack plans (based on the attack plan from the current node to the goal node). Since it is not realistic we use the measurement of alerts and similarity between alerts to evaluate the effectiveness of different honeypots. For example, if the defender will try to use such a honeypot will reduce the alert similarities between the attackers which forces the attackers to reveal more information about them.

7.4 Operational Model

Now we present the components of the operational game model. Since the capabilities of an attacker can vary due to the existence of myriad available TTP, we use a case study based operational model to show how the algorithm works. In our game model, we have constructed two attackers with different preferences and different capabilities with some similarities. The similarities between the attackers help to illustrate the importance of

using a game theoretic approach to identify the acting attacker. First, we present the set of exploits, tools all the attacker can use. Each attacker can have a subset of those capabilities with a specific preference which we define manually in such a way that highlights the effectiveness of the algorithm.

7.4.1 Capabilities and MITRE ATT&CK

All of the capabilities listed below fall into the technique category of *Exploitation of Remote Services* technique which falls into the tactics category of lateral movement according to the *MITRE ATT&CK* [26, 22]. In this technique an adversary exploits remote services to gain access to a target machine after he already has a foothold inside the network. The exploitation can occur because of remote software vulnerability which can be a bug or error in the code or even in the OS which is exploited by the attacker to his advantage. However, before the exploitation, an attacker needs to know if the remote system is vulnerable which can be known by using different network scanning tools or penetration testing tools e.g. Nmap, Metasploit [28] [24]. Below are some capabilities which can be used by an attacker to exploit a remote system. When we say capability, that means if a remote system has the corresponding vulnerability an attacker can use the capability to exploit the remote vulnerability.

JBoss JBoss capability is used by an attacker to exploit a vulnerable remote JBoss service with CVE-2007-1036. In this vulnerability, the default configuration of JBoss does not prevent access to the console and web management interfaces, which allows remote attackers to bypass authentication and gain administrative access [14]. An attacker can use the metasploit *jboss_invoke_deploy* module [21] to exploit the vulnerability.

War FTP War FTP capability is used to exploit a remote machine that is running a vulnerable service with CVE-1999-0256 [12]. In this particular vulnerability, the buffer overflow allows an attacker to execute remote code. The *warftpd_165_user* module [32] in Metasploit can be used to exploit the vulnerability.

SQLMap SQLMap [30] is an open source penetration testing tool that allows an attacker (or pen-tester) to discover SQL injection flaws and compromise a database server.

SSH This capability allows an attacker to exploit the CVE-1999-0502 [13] vulnerability using the *sshexec* Metasploit module where the target machine has default password set to null or blank for SSH. An attacker can connect with a valid user password to execute a specified payload via SSH [31].

proxychain Proxychain [18] allows an attacker to tunnel his traffic through a compromised machine's meterpreter session (to bypass firewall). This technique can be used to achieve nmap scanning or sql injection using sqlmap when the firewall is blocking the traffic from the attacker's current position. First, an attacker needs to establish a meterpreter session with a machine that has access to the target machine. Then a route is added to tunnel the traffic through the meterpreter session to the destined subnet. Then a proxy server (in the local machine) is used to listen to the traffic and route any traffic which is destined for the target subnet through the meterpreter session. After that, the attacker can easily use a proxychain tool to perform any type of remote exploit e.g. sqlmap.

Besides these capabilities, we assume that all the attackers can scan the network using Nmap.

7.4.2 Environment

We use the *Common Open Research Emulator* (CORE) [11], a network emulator to model our environment and to stage the attacker and defender interactions. Figure 7.2 shows the network we used for our operational game model. It has two legit network subnets *LegitNet1, LegitNet2*. An attacker can connect to the router *n1*. Then using his capabilities he can compromise the legit networks one after another since to reach the *LegitNet 2* an attacker has to compromise the *LegitNet 1* first. As you can see in Figure 7.2 there are two decision points. On these two points, two honeynets are connected called *Honeynet1,*

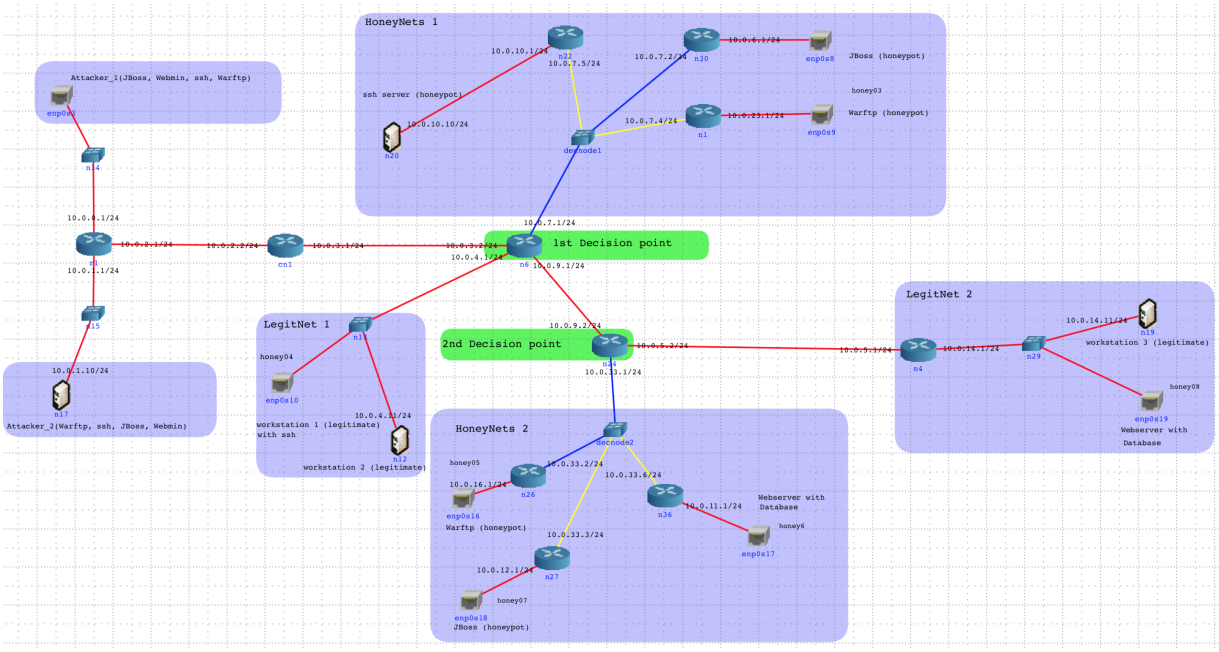


Figure 7.2: The network used for the emulated experiments

Honeynet2. The defender can activate different honeynet in those honeynets. Based on the attacker activity we assume that the defender activates only one honeynet from each of the honeynets sequentially. The objective of the defender is to lure the attacker into a honeynet which will reveal more information about his identity.

Now we describe the network topology in Figure 7.2 in detail. The network can be accessed from the outside via the 10.0.0.1/24 and 10.0.1.1/24 interface in the router *n1*. After that, there is a router *cn1*. Then comes the first decision point (near the green colored box) router *n6* which has multiple interfaces connected to it. We named this router the first decision point because we defined our operational game model as a two-step game and this is the first point where the attacker and defender have to make a decision. A legit network *LegitNet 1* is connected to the first decision point router via the 10.0.4.1/24 network interface. The *LegitNet 1* has two machines in it. A workstation *10.0.4.11/24* and a ssh server with ip *10.0.4.2/24*. The ssh server is a separate virtual machine that is connected to the network via a rj45 network interface adapter. This capability of connecting virtual

machines through rj45 port makes the network topology more realistic.

The first decision point router also has honeynets called *HoneyNets 1* connected to it in the 10.0.7.1/24 interface. The *HoneyNets 1* is connected via a decision node *decnode1* which can dynamically activate or deactivate connection towards each honeynet. We describe more about the decision nodes in a separate section where the defender algorithms are implemented. The *HoneyNets 1* has three honey networks, *ssh server*, *JBoss* and *Warftp*.

Next, we have the second decision point router *n24*. This router has a firewall that only accepts connection from the *LegitNet 1*. That means if an attacker wants to access beyond the second decision point, he has to do that by tunneling his traffic via *LegitNet 1* using a meterpreter session.

The *n24* router has e legit network *LegitNet 2* connected to it in the 10.0.5.2/24 interface. The *LegitNet 2* has workstation(10.0.14.11/24) and a webserver(10.0.14.100/24) with mysql database. An attacker's job is to compromise the webserver in the *LegitNet 2*. However, to prevent that the second decision point has *HoneyNets 2* connected to the *n24* router via the 10.0.33.1/24 interface which is connected to a decision node called *decnode2*. *HoneyNets 2* has three honey networks connected to the *decnode2*: a) Warftp(10.0.16.10/24), b) JBoss(10.0.12.2/24) c) Webserver(10.0.11.100/24) with database. So, the honeypots are in a separate network. The *decnode2* can dynamically activate/deactivate the connection to each of the honey networks in the *HoneyNets 2*. Same as before (except the ssh honeypot in HoneyNets 1) all the honeypots are connected via rj45 port which allows adding a separate virtual machine to the network.

7.4.3 Attacker Agents

We considered two attackers for our operational game model. An attacker's identity is defined by his capabilities and preferences to decide on the next attack in a sequential game. Each of the attackers has different and overlapping capabilities with each other. Same as the game theoretic model each of the attackers follows their preference of capabilities when making a decision. If there is a tie an attacker chooses the option which maximizes the

alert similarity with other attackers. Here we are also assuming that the attacker models are common knowledge.

We defined two attackers as a_{jboss} and a_{wftp}. Attacker a_{jboss} has the following capabilities with preferences: *JBoss(1), SQLMap(2), ssh(2).* Attacker a_{wftp} has the following capabilities with preferences: *warftp(1), ssh_db(2), SQLMap(2).* The number refers to their preference where a smaller number has a higher priority. We also note that both of the attackers have the *proxychain* capability besides their other capabilities where proxychain is used to exploit vulnerabilities of target machines which need to be accessed via another machine. This happens because of the firewall rules the defender has placed in the *LegitNet 2* and *HoneyNets 2* which prevents the attacker from accessing them directly. We have the scripts for the attacker capabilities upload in a GitHub repository [10].

7.5 The Game Play

The defender uses an algorithm [39] to activate different honeypots instead of activating them all at once and keeping them activated always. However, since we cannot use the theoretical algorithm straight out of the box, we transform the algorithm to make it work in the real world. The benefit of this algorithm is that it prevents the attacker from learning about the true topology of the network and also forces the attacker to reveal his identity.

As you can see in Figure 7.2, there are two different decision points (*decnoe1, decnode2*) where the defender makes his decision on which honeynet to activate. Initially all the honeynets (*HoneyNets 1, HoneyNets 2*) are deactivated from the decision points (*decnode1, decnode2*). An attacker is picked randomly between a_{jboss} and a_{wftp}. Before the attacker starts his attack the defender makes his first move by activating a honeynet in *HoneyNets 1.* The decision is made in the *decnode1* and will be described in detail in a later section. After that, the attacker decides to attack a machine which he discovers by network scanning. An attacker can use the scripts [10] to attack a machine. Since the defender activated a honeynet from the *Honeynets 1* (the attacker doesn't know whether a machine is a honeypot

or not) the attacker can either attack a machine in the *LegitNet 1* or the activated honeynet in *Honeynets 1*. If the attacker attacks a honeypot then the scenario ends. If not, then the defender strategically activates a honeynet from the *Honeynets 2*, and the decision is made by *decnode2*. And then the attacker makes his second decision. Like the previous stage if he attacks a machine in the *LegitNet 2* then he reaches his goal or he attacks a honeynet in *HoneyNets 2* then he gets caught and his identity is revealed. In our scenario, we constructed the attackers in such a way that the honeypots will always attract an attacker with its unique preference since the defender also strategically avoids using honeypots that attracts both of the attackers.

7.6 Defender Decision Engine

Three primary aspects need to be added as a part of the defender of the operational game model: a) sensing b) analysis of the traffic alerts raised by NIDS b) computing traffic pattern or alert similarity between two attackers.

Sensing The sensing aspect is a missing component in the theoretic model. In the operational model, we need the defender to have the sensing capability to have a visual on the current state of the network. Sensing can be done using different NIDS and HIDS. In our current model, we considered suricata NIDS. The rules used in the NIDS are generated by a tool called packet annotation [8] which automatically generates rules for NIDS based on supervised training. We also assume that the alerts raised by the NIDS have no error and give perfect information on an attacker's current position. However, this does not mean that the defender was able to stop the attacker. We assume that an attacker is stopped only when he gets into a honeypot.

Alert analysis The defender needs to analyze the alerts raised by the NIDS to have a visual on what is the current state of the network. The analysis of the alerts gives the defender directions on different events taking place in the network. Usually, alerts

are further analyzed by network security engineers to be investigated. However, our purpose is to use the alerts by the autonomous algorithm so that it can understand what is the current status of the attacker. I just want to remind the readers that even if the attacker is detected using NIDS doesn't mean it's been identified. The goal of our work is to use the autonomous algorithm to identify an attacker as early as possible.

Alert similarity The defender decision engine needs to compute the alert similarity between the attackers. This is one of the most crucial components of our operational model. Since in most situations an attacker cannot compute an attack path to a goal node which is more than 1 hop away due to firewall and other network security barriers, the attackers will not be able to compute attack path overlapping with one another. So, we had to transform the idea of computing attack path similarity to alert similarity. Whereas an attacker tries to increase the alert similarities with other attackers the defender wants to minimize the alert similarity. The alert similarity between two attackers can be computed by taking the summation of differences of all the features between two alerts. Since there can be multiple alerts raised by traffic activities, the feature values need to be computed for some time. We have used an already developed system which can raise alerts on attacker's current position and activity. That means we assume that our NIDS is a perfect system: the defender has a perfect observation on the attacker activities except he does not know which attacker he is currently facing.

We simplified the complexity of the traffic pattern and alert matching by assuming that the alerts are different unless the attackers use the same exploit which can happen since attackers can have similar exploits. We plan to extend the complexity in the future where there can be legitimate traffic and imperfect NIDS. Now that we have described the necessary background we present our game theoretic algorithm [39].

In each decision point, the algorithms run as a separate entity. We made it sequential;

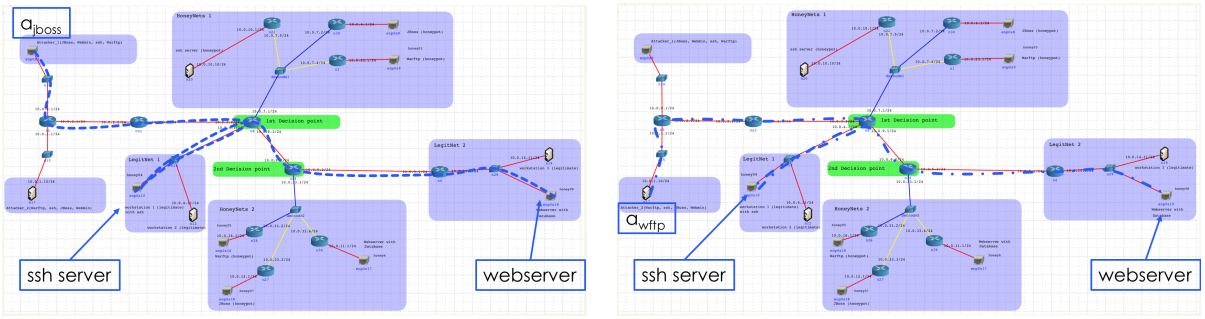
that means the *decnode1* makes a decision first, then the *decnode2* makes its decision based on the activity of the attacker in the first decision point. In the *decnode1* the algorithm computes the alert similarities between attackers for every deployment of honeynets. Then he chooses to activate the honeynet which resulted in the minimum alert similarity between the attackers. If there are two honeynet deployment settings with similar alert similarity then the defender takes into account a weighted alert similarity where the weights keep track of the success history of different honeynets. Once the decision has been made the algorithm instructs to activate the connection of that honeynet. Next, we describe the structure of the whole system and how the defender's decision engine fits into the as a module. And we describe how the decision engine itself has been implemented. The code of the decision engine is implemented as a part of the CDES [2, 3].

7.7 Results

For our experiment, we want to show that the defender can strategically deploy dynamic honeynets based on the alerts he receives to identify an attacker earlier in its attack stage. We use the CDES [] to run our experiments. We developed a case study based experiments. If we want to use the algorithm in some other network, the attackers have to be built on the existing vulnerabilities in the network. Now we move forward with the experiment using the network 7.2.

We assume that the attacker already has a foothold inside the network. As the figure shows the network has an entry point at router *n1*, where an attacker can be connected from the outside world through the 10.0.0.1/24 interface and start his post-exploitation activities. The network has two decision point *decnode1, decnoe2* which are connected to router *n6* and *n24* respectively. The router *n6* has a *LegitNet 1* and *Honeynets 1* connected to it. In both of the decision point, the defender makes the first move and then the attacker makes his move.

In our experiment, we try to show how the defender makes a strategic decision if dif-



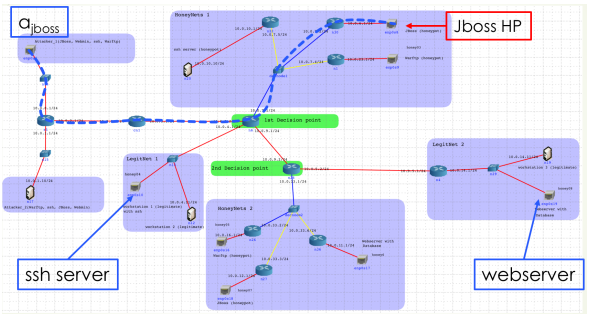
(a) a_{jboss} attack propagation without deception (b) a_{wftp} attack propagation without deception

Figure 7.3: Attack propagation without deception

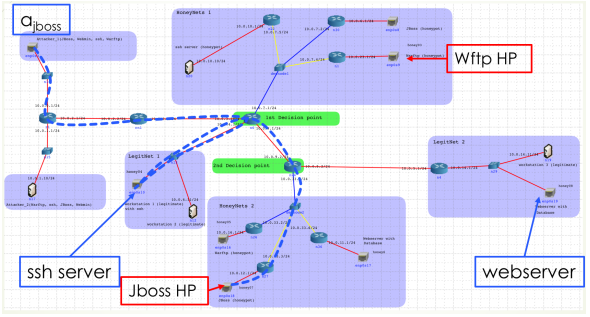
ferent attackers with overlapping capabilities attack. We focus on the defender's capability of making a strategic decision to reduce similarities between possible future alerts raised by different attackers which helps to identify an attacker. We defined two attackers as a_{jboss} and a_{wftp}. Attacker a_{jboss} has the following capabilities with preferences: *JBoss(1), SQLMap(2), ssh(2).* Attacker a_{wftp} has the following capabilities with preferences: *warftp(1), ssh_db(2), SQLMap(2).* The number refers to their preference where a smaller number has a higher priority. Both of the attacker's goal is the webserver(10.0.14.100/24) in the *LegitNet 2.*

For our first experiment, we do not use any deception strategy for attacker identification. Figure 7.3a and Figure 7.3b show the attack propagation in the network for the a_{jboss} and a_{wftp} attackers respectively. We assume that the defender ignores the IP address associated with the attacker's network traffic. We observe that both of the attackers follow the same attack plan which makes them indistinguishable.

In the *decnode1* the defender activates the *wftp* and *jboss* honeynets with uniform distribution since both of the honeypots raises same minimum alerts. The code for the decision making algorithm has uploaded in a GitHub repository [2] [3]. For both cases, as shown in Figure 7.4a 7.4b, the attacker is caught at honeypots which are earlier than the *webserver* in the *LegitNet 2.* Another observation is that we cannot use hop distance as a metric to measure the effectiveness of the deception strategies as the deployment of honeypot can

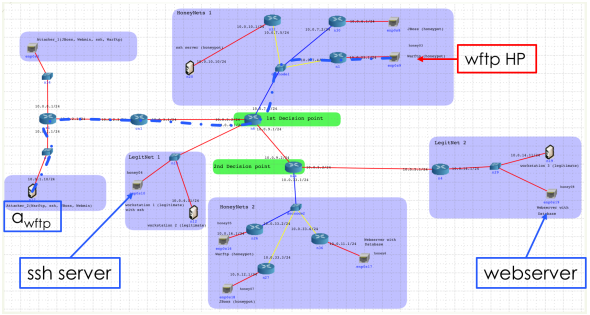


(a) a_{jboss} attack propagation with deception

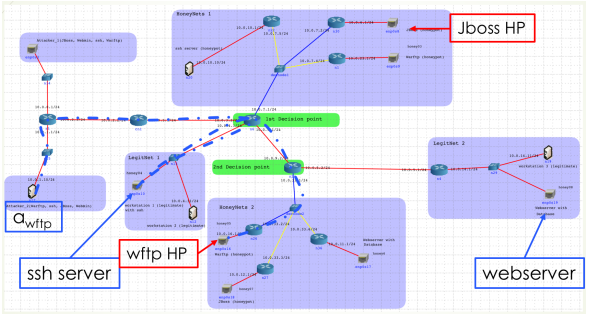


(b) a_{jboss} attack propagation with deception

Figure 7.4: Attack propagation with deception for the attacker a_{jboss}



(a) a_{wftp} attack propagation with deception

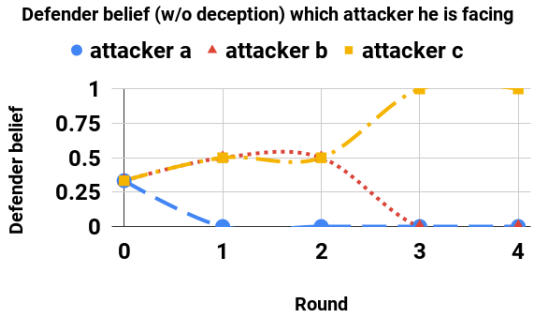


(b) a_{wftp} attack propagation with deception

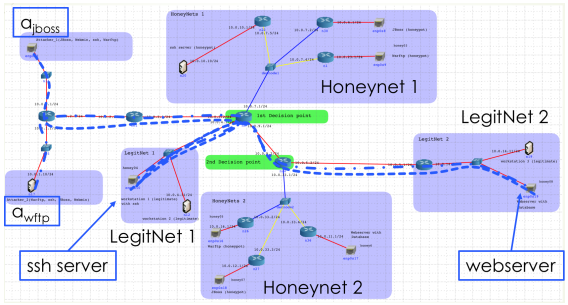
Figure 7.5: Attack propagation with deception for the attacker a_{wftp}

have longer hop distance from the attacker's current point. Similarly Figure 7.5a 7.5b show that the attacker a_{wftp} is caught at an earlier point before the *webserver* in the *LegitNet 2*.

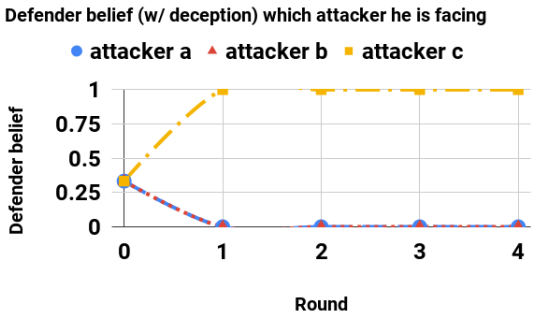
Since the operational model does not use hop distance to measure the effectiveness of strategic cyber deception, for now, we just make sure that the attacker is not able to reach his goal node *webserver* in the *LegitNet 2*. Figure 7.6a and Figure 7.6c shows the effectiveness of using strategic cyber deception technique. Figure 7.6b and Figure 7.6d shows if use of strategic cyber deception is effective. We can observe that because of strategic cyber deception, the a_{jboss} attacker was not able to reach his goal node whereas without any cyber deception the two attackers (a_{jboss}, a_{wftp}) plan become indistinguishable.



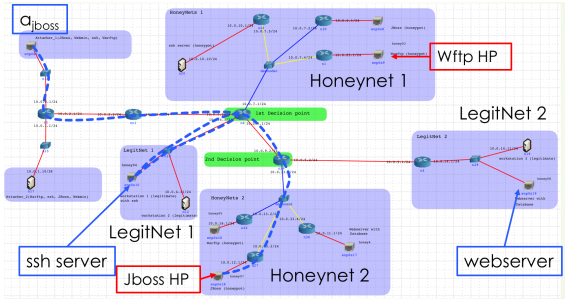
(a) Identification of attacker c without deception



(b) Indistinguishable network propagation by two attackers



(c) Identification of attacker c with deception



(d) Early identification of attacker a_{jboss}

Figure 7.6: Comparison of results between the game theoretic framework and the operational model

7.8 Conclusions

We have presented our first attempt on building an operational game theoretic model in the cyber domain where the defender strategically allocates honeynets to reveal the identity of an attacker at an early stage of its life cycle. We have taken the game theoretic model from Chapter 6 and transformed it into an operational model that works in the real world. However, it has some limitations; there is no legit traffic, the attacker is naive, the NIDS is perfect. For the future, we plan to incorporate features into the game model to handle these complexities. In the next chapter, I summarize my contributions and discuss future works.

Chapter 8

Looking Back and Moving Forward

8.1 Summary and Future Work

As I look back to when I have started my research on developing scalable algorithms in the decision making domain, I can not help but think that the scalability problem is still a very significant problem both in academia and industry. Whereas the industry tries to focus more on adhoc solutions by leveraging the state of the art technological advancements, academia focuses more on the theoretical part of the problem. I like to think that both of the throngs are important and complement each other. For example, in the industry, it's possible to utilize the quantum property of objects or physical property of an object to bypass the temporal distance between computation. The theoretical spearhead however focuses more on the algorithmic part within physical constraints. The problem is so significant that there is a framework to save the advancement of the research: P, NP problems since much past research were not able to solve the issue completely but the research needs to be preserved and utilized.

The focus of my research is more on the algorithmic part of a game-theoretic model that analyses the decision making of intelligent agents in a network environment. In this case, the algorithm needs to iterate over all of the action space. However, the action space increases exponentially to the size of the representation of the underlying network. My primary research question spearheads the idea of exploiting the characteristics of the underlying network in a game theoretic model to abstract the action space in an intelligent fashion. I present multiple research questions in different domains where the research questions are tied back to my primary research question.

The first game theoretic model I present is GSG. In this game model, the patrollers try to protect wildlife and natural resources from environmental crimes by following a patrolling path. Due to the graph representation of a large physical terrain the growth of patrolling paths increases exponentially. However, the animals in a large area are sparse and the poachers tend to be more active where there is high wildlife activity. So, I introduce the research question 1 in page 2: how can we exploit the sparsity of animal activity in the graph in GSG to achieve an abstracted game model?. I introduce a Double Oracle algorithm combined with graph contraction that incrementally adds high wildlife activity areas and patrolling paths to an abstracted game model until the attacker attacks in the abstracted game. The experiments show approximately 99.6% improvements on runtime over the baseline algorithm with an approximately 6% shift in the solution quality from the optimal solution.

The next game-theoretic model I present is based on a cyber defense scenario using an NFG where a defender uses automated NIDS/HIDS to stop a bot that propagates through a network with subnets. It is hard to solve an NFG for real-world scenarios because enumerating all possible strategies results in an extremely large game because the large number of hosts and interconnections in an enterprise network can lead to intractable NFG models. However, since I assume that the network has subnets and due to network segmentation it is difficult for a bot to spread across subnets, I answer the research question 1 in page 2 : Can we use network decomposition utilizing the subnet structure to obtain a high-quality scalable NFG?. I present an abstraction algorithm called Subgame Abstraction and Solution Concept (SASC) which utilizes the partially independent subgame structures in an NFG to achieve a highly scalable game model. I also introduced an iterative version of the algorithm (ISASC) that adjusts the noise outside the subgames of an AIOS game. The experiments show that using network decomposition the ISASC can solve huge NFG within seconds.

Next, I introduce a game-theoretic framework that represents the interaction between an APT and a defender where the objective of the APT is to reach his goal machine by

hiding his identity. The defender wants to reveal the identity of the APT and stop it by pro-actively strategically using deception. However, when the defender wants to deploy deception he needs to iterate all the honeypot deployment settings in all the places possible in the network. As a result, the action space increases exponentially with the size of the network. In this work I explore the research question 1 in page 5; whether revealing more information to the defender can reduce the action space. Our experiments show that information reveal can help to predict the next location of the attacker; and the defender can further reduce the action space to make the game model more scalable.

In the next project, I introduced different physical constraints of a realistic computer network to my last project. Due to the constraints, the previous game model needed to be transformed into a realistic one. As a result, instead of computing attack paths, the attacker utilized alert levels to make decisions. However, for the defender, the problem of iterating over action space remains. In this work, I answer the research question 1 in page 5; whether NIDS can be utilized to reduce the action space of the defender. As my first attempt, I used the packet annotation tool [] and NIDS [] to determine the attacker location that consequently reduces the action space that needs to be considered to make the strategic decision of deploying deceptions in the network.

Finally, my observation is that exploiting the characteristics of a network to reduce the action space resulted in very scalable algorithms in different domain of the game theoretic models. However, some limitations could be improved. For example, I worked only with perfect observation games and did not include any uncertainty. In the future, I would like to incorporate these features if I get the opportunity.

8.2 Future Work

The attacker identification project has three major components: 1) agent modeling 2) situational awareness 3) early detection and identification of an attacker using game-theoretic algorithms and Adversarial ML utilizing dynamic deception.

Modeling an Advanced Threat is challenging because it is a combination of human and malware. In the industry, there are very few resources to emulate an APT. Caldera is one of them which utilizes the MITRE ATT&CK database. However, it only follows a heuristic-based decision based on pre and postconditions, it does not consider defender actions into consideration. We want to build an APT emulator which can be utilized to evaluate different APT detection, identification mechanisms, and algorithms for cyber defense. We approach this problem by utilizing the field of Knowledge Representation and Reasoning (KRR) which focuses on designing computer representations of knowledge.

In the real world, in the red team and blue team exercises the blue team tries to reconstruct the timeline for the red team's attacks. If there is any gap in the attacker activity timeline due to a lack of sensor placements in the network, the network analysts manually make the decisions. For situational awareness purpose, I can use off the shelf tools like the components of Security Onion e.g. Sguil for network intrusion detection and wazuh for host intrusion detection. There is also the problem of processing a huge amount of traffic and event data. However, using the game-theoretic algorithm I am working on, it is possible to strategically activate sensors in different parts of the network based on the current alerts. This way not only early detection will happen due to honeypots but also we can reduce the load of a huge amount of data and events collected by the sensors.

The game-theoretic algorithm will observe the alerts provided by the sensors placed in the system and will strategically deploy dynamic honeypots to identify the active APT earlier. We can extend the game model in different ways. The first attempt would be to model this with a DNN to represent sequential decision making and using RL to retrain parts of the DNN. The defender can utilize the causal relation between the alerts and the attacker activities in the strategic decision making of revealing the identity of an attacker. The DNN with the RL can help with the pattern matching of traffic and strategic decision making. Another idea would be to use a Monte Carlo Search Tree to find the best decision in each round.

Chapter 9

Publication

A. Basak, C. Kamhoua, C. Kiekintveld et al., "Identifying Stealthy Attackers in a Game Theoretic Framework Using Deception" *Conference on Decision and Game Theory for Security*, 2019.

A. Basak, J. Cerny, et al., "An Initial Study of Targeted Personality Models in the FlipIt Game," *Conference on Decision and Game Theory for Security*, 2018.

A. Basak, F. Fei, M. Gutierrez, and C. Kiekintveld, "Algorithms for Subgame Abstraction with Applications to Cyber Defense," *International Workshop on Optimization in Multi-agent Systems, IJCAI*, 2018.

A. Basak, F. Fei, T. Nguyen, and C. Kiekintveld, "Combining Graph Contraction and Strategy Generation for Green Security Games," *Conference on Decision and Game Theory for Security*, 2016.

A. Basak, F. Fei, T. Nguyen, and C. Kiekintveld, "Grid Map Contraction In Green Security Games," *Security and Multi-Agent Systems, International Conference on Autonomous Agents and Multi-Agent Systems*, 2016.

A. Basak, and C. Kiekintveld, "Abstraction Using Analysis of Subgames," *AAAI-16*, 2015.

Chapter 10

Vita

Anjon Basak

EDUCATION

University Of Texas at El Paso, El Paso, Texas, USA

- Ph.D. student in Computer Science — Aug 2013 – May 2020
  - Thesis Topic: Abstraction techniques in solving large scale security games with underlying network structure.
  - Adviser: Prof. Christopher Kiekintveld
  - Research Interests: AI, game theory, decision making, predictive agent modeling, multi-agent system, planning, networks, scalable algorithms, intelligent agents, security.
  - Cumulative GPA: 3.87 / 4.00
- Msc in Computer Science — Aug 2013 – Dec 2016
  - Adviser: Prof. Christopher Kiekintveld

Bangladesh University of Engineering and Technology, Dhaka, Bangladesh — Feb 2006 – 2011

- Bachelor of Science (B.S.) in Computer Science and Engineering

ACADEMIC EMPLOYMENT

Graduate Research Associate, IASRL, University Of Texas at El Paso — Aug 2013 –

- Using graph contraction and Double Oracle algorithm to make Green Security Games scalable.
- Nash Equilibrium Computation in a Distributed Manner
- Modeling human behavior to better predict their actions in a dynamic environment and optimize resource allocation strategy for those different classes of behavior.
- Detecting and Identifying behavioral characteristics earlier using Bayesian updates and strategically employing a pro-active approach.

Researcher, Network Science Lab, Army Research Laboratory — Jul 2018 – Sep 2018

- Detecting and Identifying behavioral characteristics earlier using Bayesian updates and strategically employing a pro-active approach.

AWARDS & SCHOLARSHIPS

- 2nd best team in Tracer FIRE Cyberforensic Training from Sandia National Laboratories. 2019
- The Most Visionary Paper Award,

[1] A. Basak, F. Fei, T. Nguyen, and C. Kiekintveld, "Abstraction methods for solving graph-based security games," *Security and Multi-Agent Systems, International Conference on Autonomous Agents and Multi-Agent Systems*, . 2016

PUBLICATIONS

[1] A. Basak, C. Kamhoua, C. Kiekintveld et al., "Identifying Stealthy Attackers in a Game Theoretic Framework Using Deception" *Conference on Decision and Game Theory for Security*, . 2019

[2] A. Basak, J. Cerny, et al., "An Initial Study of Targeted Personality Models in the FlipIt Game," *Conference on Decision and Game Theory for Security*, . 2018

[3] A. Basak, J. Cerny, et al., "Differentiating Attackers In CyberSecurity," *AI and Computational Psychology: Theories, Algorithms and Applications, IJCAI*, . 2018

[4] A. Basak, F. Fei, M. Gutierrez, and C. Kiekintveld, "Algorithms for Subgame Abstraction with Applications to Cyber Defense," *International Workshop on Optimization in Multi-agent Systems, IJCAI*, . 2018

[5] A. Basak, F. Fei, T. Nguyen, and C. Kiekintveld, "Combining Graph Contraction and Strategy Generation for Green Security Games," *Conference on Decision and Game Theory for Security*, . 2016

[6] A. Basak, F. Fei, T. Nguyen, and C. Kiekintveld, "Grid Map Contraction In Green Security Games," *Security and Multi-Agent Systems, International Conference on Autonomous Agents and Multi-Agent Systems*, . 2016

[7] A. Basak, and C. Kiekintveld, "Abstraction Using Analysis of Subgames," *AAAI-16 Student Abstract and Poster*, . 2015

[8] A. Basak, and C. Kiekintveld, "Abstraction Using Analysis of Subgames," *Algorithmic Game Theory Workshop, International Joint Conference on Artificial Intelligence*, . 2015

References

[1] Advanced Persistent Threat Groups. https://www.fireeye.com/current-threats/apt-groups.html.

[2] Algorithm Implementation in the decnode1. https://github.com/ARL-UTEP-OC/cdes_apt_hunting/blob/master/cdesDecisionEngine.py.

[3] Algorithm Implementation in the decnode2. https://github.com/ARL-UTEP-OC/cdes_apt_hunting/blob/master/cdesDecisionEngine2.py.

[4] APT37(REAPER): The overlooked north korean actor. https://www2.fireeye.com/rs/848-DID-242/images/rpt_APT37.pdf. Accessed: 2019-06-03.

[5] APT38: Details on New North Korean Regime-Backed Threat Group. https://www.fireeye.com/blog/threat-research/2018/10/apt38-details-on-new-north-korean-regime-backed-threat-group.html.

[6] APT38, Un-usual Suspects. https://content.fireeye.com/apt/rpt-apt38.

[7] APT40: Examining a China-Nexus Espionage Actor. https://www.fireeye.com/blog/threat-research/2019/03/apt40-examining-a-china-nexus-espionage-actor.html. Accessed: 2019-06-03.

[8] ARL-UTEP-OC/eceld-netsys. https://github.com/ARL-UTEP-OC/eceld-netsys.

[9] Artifact Kit. https://www.cobaltstrike.com/help-artifact-kit.

[10] Attacker Scripts. https://github.com/ARL-UTEP-OC/cdes_apt_hunting/tree/master.

[11] Common Open Research Emulator (CORE). https://www.nrl.navy.mil/itd/ncs/products/core.

[12] CVE-1999-0256. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0256.

[13] CVE-1999-0502. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0502.

[14] CVE-2007-1036. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-1036.

[15] Defense Evasion. https://attack.mitre.org/tactics/TA0005/.

[16] Enterprise Tactics. https://attack.mitre.org/tactics/enterprise/.

[17] Enterprise Techniques. https://attack.mitre.org/techniques/enterprise/.

[18] How to Use Nmap with Meterpreter. https://www.blackhillsinfosec.com/use-nmap-meterpreter/.

[19] Initial Access. https://attack.mitre.org/tactics/TA0001/.

[20] Insights into Iranian Cyber Espionage: APT33. https://www.fireeye.com/blog/threat-research/2017/09/apt33-insights-into-iranian-cyber-espionage.html.

[21] JBoss DeploymentFileRepository WAR Deployment (via JMXInvokerServlet). https://www.rapid7.com/db/modules/exploit/multi/http/jboss_invoke_deploy.

[22] Lateral Movement. https://attack.mitre.org/tactics/TA0008/.

[23] M-Trends 2019 FireEye Mandiant Service, Special Report. https://content.fireeye.com/m-trends. Accessed: 2019-06-04.

[24] Metasploit. https://www.metasploit.com/.

[25] MITRE Initial Access Techniques. https://attack.mitre.org/tactics/TA0001/.

[26] MITRE Initial Access Techniques. https://attack.mitre.org/matrices/enterprise/.

[27] New Targeted Attack in the Middle East by APT34. https://www.fireeye.com/blog/threat-research/2017/12/targeted-attack-in-middle-east-by-apt34.html.

[28] Nmap. https://nmap.org/.

[29] PRE-ATT&CK Techniques. https://attack.mitre.org/techniques/pre/.

[30] SQLMap. http://sqlmap.org/.

[31] SSH User Code Execution. https://www.rapid7.com/db/modules/exploit/multi/ssh/sshexec.

[32] War-FTPD 1.65 Password Overflow. https://www.rapid7.com/db/modules/exploit/windows/ftp/warftpd_165_pass.

[33] Govt of Mozambique announces major decline in national elephant population, May 2015.

[34] The IUCN Red List of threatened species, April 2015.

[35] Estimate of global financial losses due to illegal fishing, February 2016.

[36] Tansu Alpcan and Tamer Basar. A game theoretic analysis of intrusion detection in access control systems. In *2004 43rd IEEE Conference on Decision and Control (CDC)(IEEE Cat. No. 04CH37601)*, volume 2, pages 1568–1573. IEEE, 2004.

[37] Tansu Alpcan and Tamer Basar. An intrusion detection game with limited observations. In *12th Int. Symp. on Dynamic Games and Applications, Sophia Antipolis, France*, volume 26. Citeseer, 2006.

[38] Nolan Bard, Deon Nicholas, Csaba Szepesvári, and Michael Bowling. Decision-theoretic clustering of strategies. In *AAMAS*, 2015.

[39] Anjon Basak, Charles Kamhoua, Sridhar Venkatesan, Marcus Gutierrez, Ahmed H Anwar, and Christopher Kiekintveld. Identifying stealthy attackers in a game theoretic framework using deception. In *International Conference on Decision and Game Theory for Security*, pages 21–32. Springer, 2019.

[40] Nicola Basilico, Nicola Gatti, and Francesco Amigoni. Patrolling security games: Definition and algorithms for solving large instances with single patroller and single intruder. *Artificial Intelligence*, 184:78–123, 2012.

[41] Gernot Veit Batz, Robert Geisberger, Sabine Neubauer, and Peter Sanders. Time-dependent contraction hierarchies and approximation. In *Experimental Algorithms*, pages 166–177. Springer, 2010.

[42] Michael Bloem, Tansu Alpcan, and Tamer Basar. Intrusion response as a resource allocation problem. In *Proceedings of the 45th IEEE Conference on Decision and Control*, pages 6283–6288. IEEE, 2006.

[43] Branislav Bosansky, Christopher Kiekintveld, Viliam Lisy, and Michal Pechoucek. An exact double-oracle algorithm for zero-sum extensive-form games with imperfect information. *Journal of Artificial Intelligence Research*, 51:829–866, 2014.

[44] Michael Bowling, Neil Burch, Michael Johanson, and Oskari Tammelin. Heads-up limit hold'em poker is solved. *Science*, 347(6218):145–149, 2015.

[45] Michele Breton, A Alj, and Alain Haurie. Sequential Stackelberg equilibria in two-person games. *Journal of Optimization Theory and Applications*, 59(1):71–97, 1988.

[46] Matthew Brown, William B Haskell, and Milind Tambe. Addressing scalability and robustness in security games with multiple boundedly rational adversaries. In *Decision and Game Theory for Security*, pages 23–42. Springer, 2014.

[47] Noam Brown, Sam Ganzfried, and Tuomas Sandholm. Hierarchical abstraction, distributed equilibrium computation, and post-processing, with application to a champion no-limit Texas hold'em agent. Technical report, 2014.

[48] Noam Brown and Tuomas Sandholm. Superhuman ai for heads-up no-limit poker: Libratus beats top professionals. *Science*, 359(6374):418–424, 2018.

[49] Lawrence Carin, George Cybenko, and Jeff Hughes. Cybersecurity strategies: The queries methodology. *Computer*, 41(8):20–26, 2008.

[50] Ritu Chadha, Thomas Bowen, Cho-Yu J Chiang, Yitzchak M Gottlieb, Alex Poylisher, Angello Sapello, Constantin Serban, Shridatt Sugrim, Gary Walther, Lisa M Marvel, et al. Cybervan: A cyber security virtual assured network testbed. In *IEEE Military Communications Conference (MILCOM)*, pages 1125–1130. IEEE, 2016.

[51] Zesheng Chen. *Modeling and defending against internet worm attacks*. PhD thesis, Georgia Institute of Technology, 2007.

[52] Vincent Conitzer and Tuomas Sandholm. A technique for reducing normal-form games to compute a nash equilibrium. In *AAMAS*, pages 537–544, 2006.

[53] Thomas H Cormen, Charles E Leiserson, and Ronald L Rivest. The floyd-warshall algorithm. *Introduction to Algorithms*, pages 558–565, 1990.

[54] Karel Durkota, Viliam Lisỳ, Branislav Bosanskỳ, and Christopher Kiekintveld. Optimal network security hardening using attack graph games. In *IJCAI*, pages 526–532, 2015.

[55] Alex Fabrikant, Christos Papadimitriou, and Kunal Talwar. The complexity of pure nash equilibria. In *Proceedings of the thirty-sixth annual ACM Symposium on the Theory of Computing*, pages 604–612, 2004.

[56] Fei Fang, Thanh H Nguyen, Rob Pickles, Wai Y Lam, Gopalasamy R Clements, Bo An, Amandeep Singh, Milind Tambe, and Andrew Lemieux. Deploying paws: Field optimization of the protection assistant for wildlife security. In *Proceedings of the Innovative Applications of Artificial Intelligence*, 2016.

[57] Fei Fang, Peter Stone, and Milind Tambe. When security games go green: Designing defender strategies to prevent poaching and illegal fishing. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.

[58] CR Field and RM Laws. The distribution of the larger herbivores in the Queen Elizabeth National Park, Uganda. *Journal of Applied Ecology*, pages 273–294, 1970.

[59] Sam Ganzfried and Tuomas Sandholm. Potential-aware imperfect-recall abstraction with earth mover's distance in imperfect-information games. In *Conference on Artificial Intelligence (AAAI)*, 2014.

[60] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *Experimental Algorithms*, pages 319–333. Springer, 2008.

[61] Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact routing in large road networks using contraction hierarchies. *Transportation Science*, 46(3):388–404, 2012.

[62] Andrew Gilpin and Tuomas Sandholm. A competitive texas hold'em poker player via automated abstraction and real-time equilibrium computation. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, volume 21, page 1007, 2006.

[63] Andrew Gilpin and Tuomas Sandholm. Better automated abstraction techniques for imperfect information games, with application to texas hold'em poker. In *AAMAS*, page 192, 2007.

[64] Andrew Gilpin and Tuomas Sandholm. Lossless abstraction of imperfect information games. *Journal of the ACM (JACM)*, 54(5):25, 2007.

[65] Andrew Gilpin, Tuomas Sandholm, and Troels Bjerre Sørensen. Potential-aware automated abstraction of sequential games, and holistic equilibrium analysis of texas hold'em poker. In *Proceedings of the Conference on Artificial Intelligence (AAAI)*, volume 22, page 50, 2007.

[66] Andrew Gilpin, Tuomas Sandholm, and Troels Bjerre Sørensen. A heads-up no-limit texas hold'em poker player: discretized betting models and automatically generated equilibrium-finding programs. In *AAMAS*, pages 911–918, 2008.

[67] Jacob K Goeree, Charles A Holt, and Thomas R Palfrey. Quantal response equilibrium. *The New Palgrave Dictionary of Economics. Palgrave Macmillan, Basingstoke*, 2008.

[68] Marcus Gutierrez, Jakub Cernỳ, Noam Ben-Asher, Efrat Aharonov, Anjon Basak, Branislav Bošanskỳ, Christopher Kiekintveld, and Cleotilde Gonzalez. Evaluating models of human behavior in an adversarial multi-armed bandit problem. In *41st Annual Meeting of the Cognitive Science Society (CogSci)*. CogSci, 2019.

[69] William B Haskell, Debarun Kar, Fei Fang, Milind Tambe, Sam Cheung, and Elizabeth Denicola. Robust protection of fisheries with compass. In *AAAI*, pages 2978–2983, 2014.

[70] Samid Hoda, Andrew Gilpin, Javier Pena, and Tuomas Sandholm. Smoothing techniques for computing nash equilibria of sequential games. *Mathematics of Operations Research*, 35(2):494–512, 2010.

[71] Tomas Holmern, John Muya, and Eivin Røskaft. Local law enforcement and illegal bushmeat hunting outside the serengeti national park, tanzania. *Environmental Conservation*, 34(01):55–63, 2007.

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

[73] Kyle Ingols, Richard Lippmann, and Keith Piwowarski. Practical attack graph generation for network defense. In *2006 22nd Annual Computer Security Applications Conference (ACSAC'06)*, pages 121–130. IEEE, 2006.

[74] Hiroaki Iwashita, Kotaro Ohori, Hirokazu Anai, and Atsushi Iwasaki. Simplifying urban network security games with cut-based graph contraction. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 205–213, 2016.

[75] Manish Jain, Erim Kardes, Christopher Kiekintveld, Fernando Ordónez, and Milind Tambe. Security games with arbitrary schedules: A branch and price approach. In *AAAI*, 2010.

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

[77] Jorma Jormakka and Jarmo VE Mölsä. Modelling information warfare as a game. *Journal of information warfare*, 4(2):12–25, 2005.

[78] Jan Karwowski and Jacek Mańdziuk. Stackelberg equilibrium approximation in general-sum extensive-form games with double-oracle sampling method. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent*

Systems, pages 2045–2047. International Foundation for Autonomous Agents and Multiagent Systems, 2019.

[79] 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, 2009.

[80] Christopher Kiekintveld, Viliam Lisỳ, and Radek Píbil. Game-theoretic foundations for the strategic use of honeypots in network security. In *Cyber warfare*, pages 81–101. Springer, 2015.

[81] Christian Kreibich and Jon Crowcroft. Honeycomb: creating intrusion detection signatures using honeypots. *ACM SIGCOMM computer communication review*, 34(1):51–56, 2004.

[82] Christian Kroer. *Large-Scale Sequential Imperfect-Information Game Solving: Theoretical Foundations and Practical Algorithms with Guarantees*. PhD thesis, US Army, 2018.

[83] Christian Kroer, Gabriele Farina, and Tuomas Sandholm. Smoothing method for approximate extensive-form perfect equilibrium. *arXiv preprint arXiv:1705.09326*, 2017.

[84] Christian Kroer, Gabriele Farina, and Tuomas Sandholm. Solving large sequential games with the excessive gap technique. In *Advances in Neural Information Processing Systems*, pages 870–880, 2018.

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

[86] Christian Kroer and Tuomas Sandholm. A unified framework for extensive-form game abstraction with bounds. In *Advances in Neural Information Processing Systems*, pages 615–626, 2018.

[87] Christian Kroer, Kevin Waugh, Fatma Kilinç-Karzan, and Tuomas Sandholm. Faster first-order methods for extensive-form game solving. In *Proceedings of the Sixteenth ACM Conference on Economics and Computation*, pages 817–834. ACM, 2015.

[88] Christian Kroer, Kevin Waugh, Fatma Kilinc-Karzan, and Tuomas Sandholm. Theoretical and practical advances on smoothing for extensive-form games. *arXiv preprint arXiv:1702.04849*, 2017.

[89] Marc Lanctot, Kevin Waugh, Martin Zinkevich, and Michael Bowling. Monte carlo sampling for regret minimization in extensive games. In *Advances in neural information processing systems*, pages 1078–1086, 2009.

[90] George Leitmann. On generalized stackelberg strategies. *Journal of Optimization Theory and Applications*, 26(4):637–643, 1978.

[91] Kevin Leyton-Brown and Yoav Shoham. Essentials of game theory: A concise multidisciplinary introduction. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 2(1):1–88, 2008.

[92] Peng Liu, Wanyu Zang, and Meng Yu. Incentive-based modeling and inference of attacker intent, objectives, and strategies. *ACM Transactions on Information and System Security (TISSEC)*, 8(1):78–118, 2005.

[93] Yu Liu, Cristina Comaniciu, and Hong Man. A bayesian game approach for intrusion detection in wireless ad hoc networks. In *Proceeding from the 2006 workshop on Game theory for communications and networks*, pages 4–es, 2006.

[94] Vincenzo Matta, Mario Di Mauro, and Maurizio Longo. Ddos attacks with randomized traffic innovation: botnet identification challenges and strategies. *IEEE Transactions on Information Forensics and Security*, 12(8):1844–1859, 2017.

[95] Richard D McKelvey, Andrew M McLennan, and Theodore L Turocy. Gambit: Software tools for game theory. 2006.

[96] H Brendan McMahan, Geoffrey J Gordon, and Avrim Blum. Planning in the presence of cost functions controlled by an adversary. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 536–543, 2003.

[97] Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisỳ, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356(6337):508–513, 2017.

[98] John F Nash. Equilibrium points in n-person games. *Proceedings of the national academy of sciences*, 36(1):48–49, 1950.

[99] Yu Nesterov. Excessive gap technique in nonsmooth convex minimization. *SIAM Journal on Optimization*, 16(1):235–249, 2005.

[100] Thanh H Nguyen, Francesco M Delle Fave, Debarun Kar, Aravind S Lakshminarayanan, Amulya Yadav, Milind Tambe, Noa Agmon, Andrew J Plumptre, Margaret Driciru, Fred Wanyama, et al. Making the most of our regrets: Regret-based solutions to handle payoff uncertainty and elicitation in green security games. In *Decision and Game Theory for Security*, pages 170–191. Springer, 2015.

[101] Andrew Nicholson, Tim Watson, Peter Norris, Alistair Duffy, and Roy Isbell. A taxonomy of technical attribution techniques for cyber attacks. In *European Conference on Information Warfare and Security*, page 188. Academic Conferences International Limited, 2012.

[102] Steven Noel and Sushil Jajodia. Optimal ids sensor placement and alert prioritization using attack graphs. *Journal of Network and Systems Management*, 16(3):259–275, 2008.

[103] Xinming Ou, Wayne F Boyer, and Miles A McQueen. A scalable approach to attack graph generation. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 336–345. ACM, 2006.

[104] 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 *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, pages 895–902, 2008.

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

[106] James Pita, Harish Bellamane, Manish Jain, Chris Kiekintveld, Jason Tsai, Fernando Ordóñez, and Milind Tambe. Security applications: Lessons of real-world deployment. *ACM SIGecom Exchanges*, 8(2):5, 2009.

[107] Frederic Raynal, Yann Berthier, Philippe Biondi, and Danielle Kaminsky. Honeypot forensics. In *Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop, 2004.*, pages 22–29. IEEE, 2004.

[108] Frederic Raynal, Yann Berthier, Philippe Biondi, and Danielle Kaminsky. Honeypot forensics, part ii: analyzing the compromised host. *IEEE security & privacy*, 2(5):77–80, 2004.

[109] Sankardas Roy, Charles Ellis, Sajjan Shiva, Dipankar Dasgupta, Vivek Shandilya, and Qishi Wu. A survey of game theory as applied to network security. In *2010 43rd Hawaii International Conference on System Sciences*, pages 1–10. IEEE, 2010.

[110] Tuomas Sandholm and Satinder Singh. Lossy stochastic game abstraction with bounds. In *Proceedings of the 13th ACM Conference on Electronic Commerce*, pages 880–897, 2012.

[111] Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeannette M Wing. Automated generation and analysis of attack graphs. In *Proceedings 2002 IEEE Symposium on Security and Privacy*, pages 273–284. IEEE, 2002.

[112] Eric Shieh, Bo An, Rong Yang, Milind Tambe, Craig Baldwin, Joseph DiRenzo, Ben Maule, and Garrett Meyer. Protect: A deployed game theoretic system to protect the ports of the united states. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 13–20, 2012.

[113] Eric Shieh, Manish Jain, Albert Xin Jiang, and Milind Tambe. Efficiently solving joint activity based security games. In *AAAI*, pages 346–352. AAAI Press, 2013.

[114] Yoav Shoham and Kevin Leyton-Brown. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press, 2008.

[115] S Skiena. Dijkstra's algorithm. *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica, Reading, MA: Addison-Wesley*, pages 225–227, 1990.

[116] Lance Spitzner. *Honeypots: tracking hackers*, volume 1. Addison-Wesley Reading, 2003.

[117] Oskari Tammelin, Neil Burch, Michael Johanson, and Michael Bowling. Solving heads-up limit texas hold'em. In *IJCAI*, pages 645–652, 2015.

[118] Nicholas Tsagourias. Cyber attacks, self-defence and the problem of attribution. *Journal of Conflict and Security Law*, 17(2):229–244, 2012.

[119] Jason Tsai, Christopher Kiekintveld, Fernando Ordonez, Milind Tambe, and Shyamsunder Rathi. Iris-a tool for strategic security allocation in transportation networks. 2009.

[120] Sridhar Venkatesan, Massimiliano Albanese, Ankit Shah, Rajesh Ganesan, and Sushil Jajodia. Detecting stealthy botnets in a resource-constrained environment using reinforcement learning. In *Proceedings of the 2017 Workshop on Moving Target Defense*, pages 75–85. ACM, 2017.

[121] Bernhard Von Stengel and Shmuel Zamir. Leadership with commitment to mixed strategies. 2004.

[122] David A Wheeler and Gregory N Larsen. Techniques for cyber attack attribution. Technical report, Institute for Defense Analyses, 2003.

[123] Rong Yang, Benjamin Ford, Milind Tambe, and Andrew Lemieux. Adaptive resource allocation for wildlife protection against illegal poachers. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 453–460, 2014.

[124] Rong Yang, Albert Xin Jiang, Milind Tambe, and Fernando Ordonez. Scaling-up security games with boundedly rational adversaries: A cutting-plane approach. In *IJCAI*, 2013.

[125] Zhengyu Yin, Albert Xin Jiang, Milind Tambe, Christopher Kiekintveld, Kevin Leyton-Brown, Tuomas Sandholm, and John P Sullivan. Trusts: Scheduling randomized patrols for fare inspection in transit systems using game theory. *AI Magazine*, 33(4):59, 2012.

[126] Quanyan Zhu. Game theory for cyber deception: a tutorial. In *Proceedings of the 6th Annual Symposium on Hot Topics in the Science of Security*, pages 1–3, 2019.

[127] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. In *Advances in neural information processing systems*, pages 1729–1736, 2007.

[128] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. In *Advances in neural information processing systems*, pages 1729–1736, 2008.