Czech Technical University in Prague
Faculty of Electrical Engineering

Doctoral Thesis

May 2018

Karel Durkota

Czech Technical University in Prague

Faculty of Electrical Engineering
Department of Computer Science

Game Theoretic Approach to Security Configuration Problems

Doctoral Thesis

Karel Durkota

Prague, May 2018

Ph.D. Programme: Electrical Engineering and Information Technology (P2612)
Branch of study: Information Science and Computer Engineering (2612V025)

Supervisor: prof. Dr. Michal Pěchouček, M.Sc.
Supervisor-Specialist: Mgr. Viliam Lisý, Ph.D.

Dedicated to my wife Katka, my brother Eduard and my parents. Thank you for your patience and support.

Acknowledgments

I would like to thank all my colleagues and co-authors that made this thesis possible. Special thanks belong to my supervisor specialist Viliam Lisý and my supervisor prof. Michal Pěchouček for their support through the years spent as a Ph.D. student. Many thanks also goes to Jiří Čermák, Karel Horák, Christopher Kiekintveld, and Branislav Bošanský for endless consultations and discussions about game-theoretic problems.

The work in this thesis was supported by the Czech Science Foundation (grant number 15-23235S and 18-27483Y), Office of Naval Research Global (grant number N62909-13-1-N256), Czech Ministry of Interior grant number VG20122014079, Technology Agency of Czech Republic grant number TH02010990, and Cisco Systems Inc.

Abstract

The current development of computer networks brings convenience and comfort to the users. Unfortunately, the computer networks are increasingly attacked, which poses security threats to the legitimate users. Nowadays, the network security focuses primarily on detecting the malicious activities to mitigate the attacks. These security measures may be effective in a short time, however, in a long time the adversaries adapt and learn how to overcome the security measures. Therefore, it is necessary to deploy the defense measures strategically, taking into account the attacker's adaptation in the long run.

Game theory provides strong theoretical and mathematical foundation to model adversarial decision-making problems, and algorithms to compute strategies for the involved parties. We can use game theory to model the security configuration problems, where the network administrator selects configuration actions for the network(s) that he protects, and the attacker chooses an attack plan for each of the observed network.

This thesis presents a specific game model for the security configuration problems to model two specific problems. The first problem focuses on strategically configuring and deploying honeypots (fake services used as decoys) that are used in the networks to detect and/or delay ongoing attacks. We use attack graph, a compact representation of all possible action sequences to compromise the network, to represent the attacker's set of attack plans. The second problem focuses on the data exfiltration detection in the computer networks. The network administrator configures the detectors (selects the thresholds), while the attacker chooses the amount of data to exfiltrate in each time step.

The challenges stem not only from appropriately capturing the problem essentials of the problem in the game models but also from computing the strategies of the players in such large and complex games. In both of our problems we use compact representations of the attacker's possible attack plans and develop algorithms that work over these representations. We develop novel suboptimal algorithms for both problems that scale better than the exact algorithms. We experimentally analyze the scalability of the algorithms and the quality of the produced strategies. For one of the problems we compare the optimal strategies with the human behavior and analyze the differences. Finally, we also demonstrate how game theory can be used to answer interesting and important questions to the network security researchers.

Abstrakt

Vývoj všudepřítomných počítačových sítí přináší uživatelům pohodlí. Sítě čelí neustále rostoucímu počtu útoků, které uživatele těchto sítí ohrožují. V současnosti se síťová bezpečnost soustředí hlavně na detekci a zmírnění zlomyslných aktivit. Tyto bezpečnostní techniky jsou efektivní kráktodobě, ale útočníci se přizpůsobují a učí se tato opatření překonávat. Proto je důležité nasazovat bezpečnostní opatření strategicky a brát v potaz jejich dlouhodobý efekt a přizpůsobivost útočníků.

Teorie her nabízí teoretické a matematické podklady pro modelování hráčů v rozhodovacích problémech a zároveň i algoritmy pro výpočet strategií pro zúčastněné strany. Teorie her může být použita pro modelování problému bezpečnostní konfigurace, kde síťový administrátor vybírá konfiguraci sítě, kterou brání, a útočník vybírá útočný plán, který vykoná v pozorované síti.

Tato práce představuje obecný herní model pro problémy bezpečnostních konfigurací a aplikuje ji na dva konkrétní bezpečností problémy. První problém se soustředí na strategické nastavení honeypotů (falešných cílů) v počítačových sítích pro detekci nebo zpomalení útoků na sítě. Útočník zde vybírá plán útoku podle útočného grafu, který kompaktně reprezentuje všechny známé zranitelnosti zařízení v síti. Druhý problém se soustředí na detekci exfiltrace dat v počítačových sítích. Síťový administrátor konfiguruje detektory (volí prah na množství odeslaných dat) pro každé zařízení, kdežto útočník vybírá množství exfiltrovaných dat v každém časovém okně.

Náročnost této práce spočívá nejenom ve správném zachycení podstatných aspektů problémů v herním modelu, ale také ve výpočtu hráčových strategií v komplexních herních stromech, které často rostou exponenciálně s velikostí zadaného problému. Práce využívá kompaktní reprezentace útočníkových strategií a představuje herně-teoretické algoritmy, které pracují s těmito reprezentacemi. Představujeme nové suboptimální algoritmy pro obě dvě instance problémů, které se škálují lépe než ty exaktní. Experimentálně analyzujeme škálovatelnost algoritmů a výkonnost vypočtených strategií. Dále ukazujeme, jak lze využít teorii her ke zodpovězení důležitých výzkumných otázek pro odborníky v oblasti zabezpečení počítačových sítí.

Contents

Contents **xiii**

List of Figures **xvii**

List of Tables **xix**

1 Introduction **1**
1.1 Research Goals of Thesis . . . 3
1.2 Contributions of the Thesis . . . 3
1.3 Organization of the Thesis . . . 4

2 Background **7**
2.1 Markov Decision Process . . . 7
2.1.1 Solving finite-horizon MDP - Forward Dynamic Programming . . . 9
2.2 Partially Observable Markov Decision Process . . . 9
2.2.1 Heuristic Search Value Iteration (HSVI) . . . 9
2.3 Game Theory . . . 11
2.3.1 Normal-Form Games . . . 11
2.3.2 Extensive-Form Game . . . 12
2.3.3 Solution Concepts . . . 15
2.3.4 Smallest Undominated Set (SUS) . . . 18
2.3.5 Algorithms . . . 18

3 Security Configuration Problem **27**
3.1 Game-Theoretic Model . . . 28
3.1.1 Chance Actions . . . 28
3.1.2 Defender's Actions . . . 29
3.1.3 Attacker's Information and Actions . . . 29
3.1.4 Player Utilities . . . 29
3.1.5 Game Properties and Assumptions . . . 30
3.2 Related Models . . . 33

4 Honeypot Selection Problem **37**
4.1 Introduction . . . 37
4.2 Related Work . . . 38
4.2.1 Attack Graphs . . . 38
4.2.2 Honeypots . . . 40
4.3 Attack Graph . . . 40
4.3.1 Attack Graph Updates . . . 43
4.3.2 Attack Policy . . . 44
4.4 Honeypot Selection Game Model . . . 47
4.4.1 Chance Actions . . . 47
4.4.2 Defender's Actions . . . 47
4.4.3 Players' Utilities . . . 48
4.5 Heuristic Approaches . . . 50
4.5.1 Perfect Information Heuristic (PI) . . . 50
4.5.2 Zero-Sum Heuristics (ZS) . . . 50
4.5.3 Correlated Stackelberg Equilibrium Heuristic (CSE) . . . 51
4.6 Algorithms . . . 52
4.6.1 Computing Optimal Attack Policy for Single Network using MDP . 52
4.6.2 Optimal Attack Policy for Multiple Networks in Information Set . . 59
4.6.3 Solving Games . . . 65
4.7 Experiments . . . 68
4.7.1 Experimental Setting . . . 68
4.7.2 Optimal Attack Planning . . . 69
4.7.3 Game Theoretic Algorithms Comparison . . . 72
4.7.4 Essential Network Security and Honeypot Related Analysis . . . 76
4.7.5 Case Study . . . 80
4.7.6 Respondent Behaviors . . . 81
4.7.7 Survey Analysis . . . 82
4.8 Conclusion . . . 83

5 Threshold Selection Problem **85**
5.1 Introduction . . . 85
5.2 Related Work . . . 87
5.3 Game Theoretic Model . . . 88
5.3.1 Outsider vs. Insider . . . 89
5.3.2 Additivity vs. Replacement . . . 89
5.3.3 Formal Definition of Game Model . . . 89
5.4 Algorithms . . . 91
5.4.1 Optimal Defense Strategy Against Insiders . . . 91
5.4.2 Optimal Defense Strategy Against External Attacker . . . 92
5.5 Real-world Data . . . 95
5.6 Experiments . . . 96
5.6.1 Defender and Attacker Utilities . . . 97
5.6.2 Defender's and Attacker's Strategies . . . 98

5.6.3 Different Attacker Models . . . 101
5.6.4 Effect of the Additivity . . . 102
5.6.5 Algorithm Scalability . . . 103
5.7 Conclusion . . . 104

6 Conclusions **107**
6.1 Thesis Contributions . . . 108
6.1.1 Game Models for Two Security Configuration Problem . . . 108
6.1.2 Set of Novel Algorithms . . . 109
6.1.3 Experimental Evaluations and Analyses . . . 109
6.1.4 Implementation . . . 109
6.2 Future Work . . . 109
6.3 Publications . . . 110
6.3.1 Publications Related to the Thesis . . . 111
6.3.2 Publications Not Related with the Thesis . . . 111

Bibliography **113**

A Notation **121**

List of Figures

2.1 HSVI algorithm . . . 10
2.2 Simple extensive-form game . . . 14

3.1 General game model . . . 28
3.2 EFG with unstable multi-defender solution profile . . . 33
3.3 Graphical games . . . 34

4.1 Simple attack graph . . . 42
4.2 Use example of attack graph rules . . . 45
4.3 An optimal attack policy . . . 46
4.4 Honeypot selection game model . . . 48
4.5 Proof of sibling class theorem . . . 55
4.6 Sibling-class theorem counterexample . . . 57
4.7 Three types of cycles in attack graphs . . . 59
4.8 Attacker's POMDP consisting of two MDPs . . . 60
4.9 Removal of ϵ-dominated strategies . . . 64
4.10 Illustration of LP for CSE heuristic . . . 68
4.11 Neworks used in the experiments . . . 70
4.12 Evaluation of attack graph pruning techniques . . . 71
4.13 Runtime comparisons of heuristics . . . 73
4.14 Comparison of strategy qualities from heuristics . . . 74
4.15 Defender's regret for estimating attack graphs . . . 76
4.16 Defender's utility components for different number of honeypots . . . 80
4.17 Cluster's of respondents' behaviors . . . 81

5.1 Adversarial HSVI algorithm . . . 94
5.2 Mean upload size histograms of the identified clusters of users. . . . 96
5.3 Defender's strategies against insiders in TSP . . . 99
5.4 Defender's strategies against outsiders in TSP . . . 100
5.5 Defender's strategies for users with behavior of normal distribution . . . 102
5.6 The attacker's utility given the user's behavior uncertainty . . . 102
5.8 Algorithm runtime for different problem parameters . . . 103
5.7 Algorithm runtimes for different number of host-types (clusters) . . . 103
5.9 Defender's errors for different number of types . . . 104

List of Tables

2.1 An example of simple normal-form game. 12

4.1 Optimal attack graph algorithm comparison with domain-independent solvers 71
4.2 Defender's relative regret for not using honeypots 77
4.3 Defender's regret for playing PI strategy instead of MILP 78
4.4 Attacker's entropy in information sets for different defense strategies 79
4.5 Defenders loss for assuming imperfect information attacker 79
4.6 Expected utilities of respondents' strategies 81

5.1 Attacker's utilities for different defense strategies 97
5.2 Defender's utilities for different attack models in TSP 101

A.1 Used notation 121
A.2 List of used abbreviations 122

Chapter 1

Introduction

Computer systems play a critical role in daily life, commercial interests, and military operations. System security configuration is crucial to protect the computer systems against sophisticated intruders including both criminal organizations and state actors is one of the most important challenges for national security. Network administrators often face challenging decisions in how to assess the effectiveness of security configurations, and how to optimize the way these configurations are selected and deployed for several reasons. The networks complexity is constantly increasing which dramatically increases the defenders' possible configuration space to consider. The automated tools of the adversaries become more sophisticated and strategic in compromising the networks. Defenses that appear to be effective in the short term may become less effective as strategic attackers learn and adapt to the latest security policies.

Although there is an increasing volume of research in computer network security, the main focus of the research is on specific technical methods and identification of vulnerabilities, and the suggested approaches often lack rigorous foundations and detailed modeling of the adversarial interaction between defenders and attackers. This makes it difficult to compare the effectiveness of different security measures and to select the most suitable combination of security measures to protect a computer system or network. The lack of the rigorous foundations is mostly caused by the complexity of the domain and its openness – in real-world conditions, there are endless options available to the attackers that can never fully fit into one formal model. In addition, computational scalability of solution methods is a significant problem for the existing formal models, which often cannot scale to analyzing realistic networks. In spite of these findings, it is still important to create formal models of specific situations in the network security and seek for theoretically sound solutions with well-defined performance metrics.

Algorithmic game theory provides an important theoretical framework for modeling the interaction between two or more rational decision-makers (players) in a shared environment. Game theory models the actions, that the players can perform as well as the (limited) information that each player has about the state of the world, which allows explicit reasoning about the information manipulation in adversarial settings, including deception and the use of randomized strategies. Game-theoretic algorithms have been shown to outperform people

in specialized tasks, such as winning in the chess game and recently in go ([Silver et al., 2017]) or poker ([Moravčík et al., 2017]). Game theory has also been applied in many physical security scenarios, such as deploying checkpoints at LAX international airport ([Pita et al., 2008]), scheduling air marshals to flights over US ([Tsai et al., 2009]), securing the maritime transit ([Vaněk et al., 2010]), catching the poachers ([Fang et al., 2017]), or protecting computer networks ([Mitchell and Healy, 2018]).

We use game theory to model the security configuration problem, where the defender selects system configuration actions such as configuring intrusion detections and filters. The attackers selects an action from a wide range of attack actions such as scanning the network, exploiting vulnerabilities or exfiltrate valuable data. Different security policies and configurations have varying costs and effectiveness against specific types of attacks. The defenders aim to find a strategy with the best trade-off between investing the limited resources into the security measures and their contribution that meets the possible constraints (e.g., maximal false-positives rate) and protects the company assets against the rational attackers.

In this thesis, we present two game-theoretic models for two different security configuration problems. In the *honeypot selection problem*, we focus on strategically configuring the honeypots that are deployed into the networks protected by the defender. In this model, the defender chooses a set of honeypots and their configurations, while the attacker selects attack plans to attack the networks. The set of plans for the attacker we represent using *attack graphs* that compactly represent a rich space of sequential attacks for compromising a specific computer network. They can be automatically generated based on known vulnerability databases. As far as we know, we are the first to combine the complex attack graphs with game theory to compute the defense strategies. The only previous work known to us is in [Bistarelli et al., 2006a] that uses defense trees ([Bistarelli et al., 2006b]) to decide what defense actions should the defender perform. However, in their game the player's have perfect information about the network states and the defense trees are less complex and are created manually.

The second problem focuses on data exfiltration problem, specifically, on *threshold selection problem* for the detectors. The network administrator selects the thresholds on the total amount of data, that the hosts in the network can upload, while the attacker chooses the amount of data to exfiltrate in each time step. We also analyze the impact of the attacker's ability to influence the user's standard traffic at the host during the exfiltration period on the optimal defense strategies. The model is experimentally analyzed on real-world data from large enterprise with 5864 users connecting to a Google drive service for 12 weeks.

One of the main contributions of the thesis is the game models for the above-mentioned problems. Additionally, we present a more general model that captures the essentials of the security configuration problems that can be used to model other similar configuration problems. The model is sufficient to capture the attacker's uncertainty of the attacked system as well as the defender's information manipulation for the attacker. We analyze the properties and limitations of the general game model.

Computing the solutions to specific game models has a great value for the researchers and security companies. It allows not only finding the optimal strategies to configure the

systems to mitigate the attacks but provides a quantitative measure of security and allows analyzing the shortcomings of the currently deployed non-strategic defenses.

1.1 Research Goals of Thesis

Based on the motivations mentioned earlier, we state the research questions:

What game structure is appropriate for system configuration problem? We analyze how to represent in a game the information asymmetry of the problem, players' prior beliefs, and uncertainties. We further put several key requirements that our model should meet to be realistic. (i) It should inherently model the attacker's uncertainty about an attacked system and its configuration, while the defender knows both. (ii) We require Kirchhoff's principle that the attacker may have access to all the algorithms that the defender uses as well, which avoids the security through obscurity. (iii) We require that our model explicitly captures the player's information, reasoning, and possible deception.

What algorithms can be used to solve our game models? Game theory provides an extensive library of algorithms for computing the desired solution concepts, from the most general games to specific games. Can we use existing algorithms to solve our game? What heuristic or approximation approaches are appropriate to solve larger games faster at the cost of loosing the optimality?

How to include compact representation of attacks into the games? Attacker's complex space of plans is usually represented using a compact data structure, such as attack graphs or parameters of a Markov decision process. Games often require listing all available actions of the players. The question is, whether it is possible to solve the game without listing the attacker's often exponential set of actions for the game.

How game theory can be useful for network security researchers? We use game theory to answer various questions, e.g., what happens if some of the assumptions are not met, or what is the defender's utility loss for optimizing the strategy against different attack model that the one according which the real attacker attacks. Game theory is suitable to answer these and other exciting questions.

1.2 Contributions of the Thesis

Game models for security configuration problem. We introduce a specific game model to capture the interaction between a defender who configures a set of systems, and an attacker who attacks them based on a complex decision-making process. Our model assumes realistic properties, such as (i) that the attacker has uncertainty about the possible systems that he attacks. (ii) The game explicitly models the information and reasoning of each player during the game and allows the information manipulation of the defender to possible increase the attacker's uncertainty about the state of the attacked system. (iii) Finally, we assume Kirchhoff's principle, assuming that the settings of a system (the game, the algorithms, and even the defense strategy) are *known* to the attacker.

Game models for two specific security problems. We present two novel game models for two current network configuration problems. (i) The first problem focuses on *honeypot selection problem* for the network administrator. Deciding how to optimally configure honeypots is a difficult task because it requires estimating how rational attacker can react to these honeypots and how they reason about the observed information about the networks. (ii) The second demonstration applies our game model to *threshold selection problem* for data exfiltration detectors. Here, the network administrator chooses thresholds on the amount of data-upload for the hosts with a false positive rate constraint.

We use these models to compute the optimal and near-optimal defense strategies and experimentally evaluate their performance. We show how game theory can be helpful to answer various research questions, such as impacts of incorrectly assuming certain properties. Being able to solve these games, allows us to improve the defense against the adversary by concluding how the network administrators should behave, by comparing the current strategies to optimal defense strategies and analyzing for the shortcomings in the strategies.

For the honeypot selection problem, we additionally conducted a survey to learn the humans' behavior in a simplified version of this problem and analyze the differences between their strategies and the optimal defense strategies according to the game theory.

Algorithms to compute the attacker's optimal plans. One of our main contributions is advancing the algorithms to solve the presented games. For honeypot selection problem, (i) we present novel algorithms that compute the attacker's optimal contingency attack policies, that describe how to optimally attack the networks. The algorithm formulates the problem as finite-horizon partially observable Markov decision process and uses search algorithms with various optimization techniques to compute the optimal attack policy. To solve the game, (ii) we develop set of heuristic algorithms to compute defense strategies, that are based on more restrictive game assumptions, such as zero-sum utilities, the attacker having perfect information about the network. In the threshold selection problem, the attacker is represented with infinite-horizon partially observable Markov decision process, for which (iii) we extend the Heuristic Search Value Iteration algorithm to compute and iteratively improving defender's strategy. For all algorithms we quantitatively compare the scalability of the algorithms and analyze the performance of the computed strategies. We show, that game theoretic algorithms often outperform baseline strategies.

1.3 Organization of the Thesis

The thesis is organized as follows:

- In Chapter 2 we present technical background, such as Markov decision processes and game theoretic definitions and algorithms, that we use throughout the thesis.
- In Chapter 3 we present a specific game model that captures the essentials of the security configuration problem. We extend this model to two specific security config-

uration problems. We analyze and discuss the game model properties and assumptions.

- Chapter 4 states Honeypot Configuration Problem and presents game model to solve the problem. Presents the concept of attack graphs that compactly represent the library of possible attack plans for the attacker. The optimal, error-bounded and several heuristic algorithms that compute the defense strategies are described. The experimental section experimentally evaluates the algorithms, answer various research questions about the honeypots and model assumptions. Finally, the game-theoretic strategies are compared to human strategies learned from the conducted survey.
- Chapter 5 present the threshold configuration problem and a game model for this problem. Two attack models are discussed and algorithms to compute the defense strategies against these attack models are presented. In the experimental section, the algorithms are experimentally evaluated using real-world user data.
- Finally, in Chapter 6 we conclude the thesis with the summary of the work, list of contributions, authors published, and the list future possible directions. In Appendices, the reader can find list of notations and abbreviations used in the thesis.

Chapter 2

Background

In this chapter, we introduce fundamental concepts, mathematical frameworks and algorithms for decision-making problems used across this thesis. In the first part, we focus on single-agent decision problems in an uncertain environment. In the second part, we extend our focus to problems with multiple agents in a common environment. These problems are more complex, since each agent must consider what the other agent(s) might do in reaction on the actions of the first agent to optimize his respective expected utility.

2.1 Markov Decision Process

A Markov Decision Process (MDP) is a general framework to model a decision-making problem for a single agent in stochastic environment. It is a *state-based* description of the environment, where the agent has a set of possible *actions* to perform in each state. After an action is performed a *stochastic event* occurs, which chooses a subsequent agent's state according to a known probability distribution to the agent. The agent's motivation is described with a *reward function*, which specifies the reward the agents receives for transitioning from one state to another using given action.

Definition 1 (Markov Decision Process (MDP))**.** Markov Decision Process *is defined as a tuple* $\langle S, A, T, R \rangle$*, where:*

- S *is a finite set of states,*
- A *is a finite set of actions*
- $T[s'|a, s]$ *is a stochastic transition function specifying the probability of ending in state* s' *if action* a *is played in state* s*, and*
- $R(s, a, s')$ *is the immediate reward function that the agent obtains if action* a *is played in state* s *and state* s' *is reached.*

The MDP algorithms aim to compute the *optimal policy* for the agent given a problem formalized using MDP. The optimal policy prescribes an action for each state that the agent to perform to maximize his objective function.

Definition 2 (MDP Policy)**.** *Given an MDP* $\langle S, A, T, R \rangle$*, we define* policy $\pi : S \to A$ *as a mapping from states to actions.*

Specifying an initial state $s_0 \in S$, where the agent performs his first action, applying policy π proceeds as follows. In s_0 the agent performs action $a_0 = \pi(s_0)$ and with probability $T[s_1|s_0, a_0]$ reaches state s_1 and receives immediate reward of $r_0 = R(s_0, a_0, s_1)$. If s_1 is reached, then the agent performs action $a_1 = \pi(s_1)$, reaches state s_2 with probability $T[s_2, s_1, a_2]$ and obtains $r_1 = R(s_1, a_1, s_2)$. And the whole process repeats in a similar fashion. In reality, only one sequence of states will be realized, but to compute the optimal policy all possible contingencies must be considered.

There are several objective functions typically considered in the literature that the agent may maximize. An MDP where the agent can perform only H consecutive actions, after which the MDP terminates, is called *finite-horizon* MDP. Here, the agent typically maximizes the expected reward over the horizon of H steps, computed as

$$\mathbb{E}[\sum_{t=0}^{H} r_t], \tag{2.1}$$

where r_t is the reward at time t. In the *infinite-horizon* case, where the agents has no limit on the number of performed actions, the future rewards are typically discounted by a *discount-factor* $\gamma \in [0, 1)$. The agents objective function that he maximizes is then

$$\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t]. \tag{2.2}$$

Discounting future rewards has several justifications. First, from the economic perspective it supports phrase "a dollar today is worth more than a dollar tomorrow", which states that an good investment today will augment the invested value in the future. Second, the far future is more unpredictable and discount captures a small chance that the new circumstances will prevent the agent to reach the reward. For small discount γ, the agent is mainly concerned about immediate rewards, while for γ close to one, the agent is motivated to maximize also the future rewards.

Let us introduce a *value function*. For a given state and a given policy it returns the expected reward that the agent obtains if that policy is applied in that state. For infinite-horizon MDP, the value function is defined as follows:

Definition 3 (Value function)**.** *A* value function*, denoted as $V^\pi : S \to \mathbb{R}$ is the expected utility when starting in state s and following policy π thereafter. For infinite-horizon, the function is defined recursively as*

$$V^\pi(s) = \sum_{s'} T[s'|\pi(s)s,](R(s, \pi(s), s') + \gamma V^\pi(s')). \tag{2.3}$$

For finite-horizon case, the value function requires a small modification: (i) discount factor $\gamma = 1$, and (ii) we track the number of performed actions not to exceed the horizon limit of H steps.

For given MDP instance problem and initial state s_0, the MDP algorithms aim to compute the optimal policy π with the highest value $V^\pi(s_0)$. The algorithms often rely on *Markovian assumption*, which states that the probability distribution of the stochastic events depends solely on the state, in which the agent chooses his next action to perform, and not on the full history of the states that led to the current state.

2.1.1 Solving finite-horizon MDP - Forward Dynamic Programming

Finite-horizon MDPs can be solved using basic *forward dynamic programming* (FDP) ([Bertsekas et al., 1995]) algorithm. It is a variant of depth-first search, which traverses the state-space from the initial state $s_0 \in S$ forward. In every state $s \in S$ it evaluates the expected reward of playing each action $a \in A$ after which the optimal policy is assumed to be played. After all actions in a state are evaluated, the optimal policy chooses an action with the highest expected reward.

During the MDP state space traversal, the same state may be reach via different paths. To avoid computing the value function of the same state multiple times, a *caching* (or *memoization*, *dynamic programming*) technique can be used to remember the reward of evaluated states in the past for future uses. Another speed-up technique uses fast lower and upper bound computations for actions in a state. If the upper bound of one action a is lower than the lower bound of action b, then action a can be pruned out, since it will definitely not be the optimal action to perform.

This algorithm finds optimal policy and always terminates in finite number of steps. It is so, because player has finite number of actions in each state and only finite-horizon MDP is considered. We present this algorithm in detail in Section 4.6.1 on a specific MDP problem and develop heuristic functions to improve its speed.

2.2 Partially Observable Markov Decision Process

Partially observable MDP (POMDP) generalizes MDP by disallowing the agent to directly observe the underlying state. The agent maintains a probability distribution over the set of states based on a set of observations and observation probabilities. There are variety of applications of the POMDP in artificial intelligence [Kaelbling et al., 1998, Cassandra, 1998].

Definition 4 (POMDP (from [Smith and Simmons, 2004]))**.** Partially Observable Markov Decision Process *(POMDP) is defined as* (S, A, T, R, O)*, where* S, A, T, R *are the same as in MDP and* $O[o|s', a]$ *is a stochastic observation function that specifies the probability of receiving observation* o *if action* a *is played and state* s' *is reached.*

The agent's probability distribution over the set of possible states is called *agents belief*, denoted as $b \in \Delta(S) = \mathcal{B}$, where $\mathcal{B}$ denotes full *belief-space*. When action a is performed in b, the agent observes o and updates his belief to b' using Bayesian rule:

$$b'(s') = \frac{1}{\eta} O[o|s', a] \sum_{s \in S} T[s'|s, a] b(s), \tag{2.4}$$

where $\eta = \sum_{s' \in S} O[o|s', a] \sum_{s \in S} T[s'|s, a] b(s)$. Since in POMDP the agent may not be certain of the current state, the policy prescription maps to each belief an action $\pi : \mathcal{B} \to A$.

2.2.1 Heuristic Search Value Iteration (HSVI)

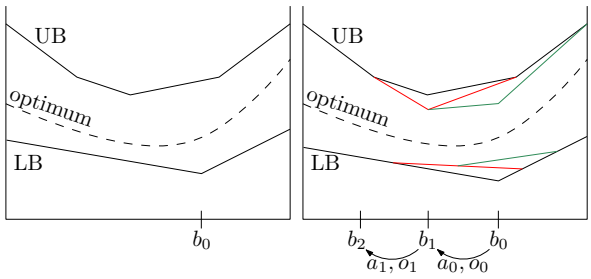
In this thesis, we use a well established Heuristic Search Value Iteration (HSVI) [Smith and Simmons, 2004] algorithm that finds ϵ-optimal policy in discounted infinite-horizon

POMDP. The algorithm maximizes value function $V^* : \mathcal{B} \to \mathbb{R}$ and approximates corresponding ϵ-optimal policy with maximal ϵ error in initial belief $b_0 \in \mathcal{B}$ of given POMDP. We can then derive the optimal action to play in each belief state, i.e. the action $\pi(b)$, by solving the following equation

$$\pi(b) = \underset{a \in A}{\operatorname{argmax}} \left[\sum_{s \in S} \sum_{s' \in S} T[s'|a, s] b(s) R(s, a, s') + \gamma \sum_{o \in O} O[o|b, a] \cdot V^*(\tau(b, a, o)) \right] \tag{2.5}$$

where V^* is the value of the optimal policy and $\tau(b, a, o)$ stands for a Bayesian update of the belief b based on receiving the observation o when action a was performed as described in Equation 2.4.

Here we describe the basic idea of the HSVI algorithm, which we complement with illustrations in Figure 2.1. For more detailed explanation of the HSVI algorithm, we refer the reader to [Smith and Simmons, 2004]. The algorithm maintains the upper and lower bounds on the optimal value function V^* for each point in the belief space $\mathcal{B}$, as depicted in Figure 2.1a. The horizontal axis represents the belief space $\mathcal{B}$ and the vertical axis represents the expected utility the agent can achieve (or lower and upper bounds on this utility, respectively). In each iteration, HSVI performs a single simulation, starting from the initial belief b_0, in which the agent performs certain actions and certain obtains observations. The actions and observations and chosen according to a forward-exploration heuristic, which aims to maximally decrease the lower and upper bound gap in the initial belief state b_0. Each simulation results in a sequence of belief states, actions and observations for the agent, as depicted in Figure 2.1b. During the simulation backtrack, the agent updates the lower bound and upper bounds on V^* in the visited beliefs according to Equation 2.5. In Figure 2.1b we illustrate an example backtrack during which the lower and upper bounds of b_2 are used to improve the bounds of b_1 (red colored lines), and those are used to improve the bounds of b_0 (green colored lines). Every iteration the error gap in b_0 can only diminish



(a) (b)

Figure 2.1: HSVI algorithm: (a) initial upper bound (UB) and lower bound (LB) on the (unknown) optimal value function V^*. HSVI aims to minimizes the gap between UB and LB in the initial belief b_0. (b) Shows a simulation into depth 2, after which tighter bounds of LB and UB are computed.

and when the error gap is less than or equal to ϵ, the algorithm terminates with ϵ-optimal policy.

The outline of the HSVI algorithm is presented in Algorithm 1. First, it initializes the upper and lower bounds. The lower bound is computed by fixing the policy to play always action a and chooses such a, that their expected reward is maximized ([Hauskrecht, 1997]). The upper bound is computed by assuming the agent's full observability of the states, which turns to be a MDP ([Astrom, 1965]). Then, the algorithm calls exploration function in Algorithm 2, which simulates one run of the POMDP into a such depth, that the error gap weighted by the discount factor $\gamma^t(UB(b) - LB(b))$ is smaller than the desired ϵ error. During the backtracking from depth t, the explore function updates the upper and lower bounds and decreases their gap at every reached belief. The updating functions are more complex and we direct the readers to see the original paper [Smith and Simmons, 2004] for more details.

Algorithm 1 HSVI algorithm.

```
1: procedure HSVI(ε)
2:     Initialize the bounds UB and LB
3:     while UB(b_0) − LB(b_0) > ε do
4:         EXPLORE(b_0, ε, 0)
5:     end while
6:     return π
7: end procedure
```

Algorithm 2 Explore subroutine.

```
1: procedure EXPLORE(b, ε, t)
2:     if γ^t(UB(b) − LB(b)) ≤ ε then
3:         return
4:     end if
5:     select a*, o* according to the forward exploration heuristics
6:     EXPLORE(τ(b, a*, o*), ε, t + 1)
7:     update bounds UB and LB at b
8: end procedure
```

2.3 Game Theory

So far we focused on optimal decision-making of a single agent in a stochastic environment. However, the environment may contain other decision-makers, perhaps even adversarial, that *react* on the actions carried out by the agent. Game theory is a mathematical framework to model the interaction between two or more independent, self-interested agents (players) in a common environment. It provides fundamental concepts, guarantees and algorithms that aim to compute a description of the optimal behavior of the players in a shared environment with limited information. A game consists of (i) *players*, (ii) *strategies* (actions) available to each player, and (iii) *utilities* for each player depending on the joint choices of all players.

2.3.1 Normal-Form Games

One-shot games where the players perform their actions simultaneously are called *normal-form games* (NFG). In this thesis we focus primarily on games with two players, defined as follows:

Table 2.1: An example of simple normal-form game.

	Left	Right
Up	1,2	3,0
Down	0,0	2,1

Definition 5 (Normal-Form Game (NFG))**.** *A* normal-form game *is a tuple* $(\mathcal{N}, \mathcal{A}, u)$*, where:*

- $\mathcal{N} = \{1, 2\}$ *is a set of* players*. We also use* i *to refer to one player and* $-i$ *to refer to the opponent of* i*,*
- $\mathcal{A}_i$ *is a finite sets of* actions *available to player* i*,*
- $u_i : \mathcal{A}_i \times \mathcal{A}_{-i} \to \mathbb{R}$ *is player* i*'s real-valued utility for each pair of actions of the players. Each player* i *maximizes his utility.*

Each player i chooses an action $a_i \in \mathcal{A}_i$. Two-player NFGs are sometimes called *bimatrix games* since they can be represented as a matrix where player 1 choose a row of the matrix and player 2 chooses a column of the matrix. Such example can be found in Table 2.1. Here, player one chooses between actions Up or Down, while player 2 chooses between actions Left and Right. Each cell presents the payoff that player 1 and 2 obtain, respectively, if that outcome is reached. E.g., if player 1 plays Up and player 2 plays Left, they receive utility 1 and 2, respectively. In NFG the actions are termed *pure strategies*, denoted as $\Pi = \Pi_1 \times \Pi_2$, where $\Pi_i = \mathcal{A}_i$.

We define *mixed strategies* for player i as a probability distribution $\delta_i \in \Delta_i = \Delta(\Pi_i)$ over the pure strategies. A *solution profile* is a pair of strategies (δ_i, δ_{-i}), one for each player, where $\delta_i \in \Delta_i$, for each $i \in \mathcal{N}$. The *expected utility* of player i in solution profile (δ_i, δ_{-i}) is $u_i(\delta_i, \delta_{-i}) = \sum_{a_i \in \mathcal{A}_i} \sum_{a_{-i} \in \mathcal{A}_{-i}} \delta_i(a_i)\delta_{-i}(a_{-i})u_i(a_i, a_{-i})$.

2.3.2 Extensive-Form Game

Now consider a game of chess, where players must act sequentially instead of simultaneously. Sequential games can be represented as a matrix game, where an action of a player states full contingent behavior of the player. In other words, each action (a row or a column) in a matrix game contains a reactions of the player to all reachable contingencies in the chess game. Matrix payoffs in such game can be computed by simulating of the pair of (contingency) actions: a contingency action of the row player against a contingency action of a column player, and determining which player has won. Obviously, such game representation is inefficient and will result in enormously large matrix game, but conceptually it is possible to capture any sequential game with NFG.

A more compact representation offers an *extensive-form game*, which inherently captures the sequentiality of the players. These games can be visualized by a game tree, where each node is assigned to a player who acts in that node, as depicted in Figure 2.2a. Here, each player chooses an action, represented by an edge, to play in each of his decision node.

Definition 6 (Extensive-form game (EFG))**.** *An* imperfect-information game *is a tuple* $(\mathcal{N}, \mathcal{A}, \mathcal{H}, \mathcal{Z}, \mathcal{P}, u, \mathcal{C}, \mathcal{I})$*, where:*

- $\mathcal{N} = \{1, 2\}$ *is a set of players;*
- H *is a finite set of choice* nodes *in the game. Each node corresponds to a unique* history *of actions taken by all players and Nature from he root of the game. Hence, we use the terms history and node interchangeably;*
- $\mathcal{Z} \subseteq \mathcal{H}$ *is a set of* terminal *(or* leaf*) nodes;*
- $\mathcal{A}_i$ *denotes a set of all actions of player* i*. We overload the notation and use* $\mathcal{A}_i(h) \subseteq \mathcal{A}_i$ *to represent the set of actions available to player* i *in node* $h \in H$*;*
- $u_i : \mathcal{Z} \to \mathbb{R}$ *is a* utility function *for each player* i*. Each player maximizes his utility function;*
- $\mathcal{P} : \mathcal{H} \to \mathcal{N} \cup \{c\}$ *assigns each node to a player who takes an action in the node, where* c *means that the* Chance *(or* Nature*) player selects an action in the node based on a fixed probability distribution known to all players;*
- $\mathcal{C} : \mathcal{H} \to [0, 1]$ *denotes the probability of reaching node* h *due to the Chance player. It assumes, that all other players play all the required actions to reach node* h*. The value* $\mathcal{C}(h)$ *is the product of the probabilities assigned to all actions taken by the Chance player in history* h*;*
- $\mathcal{I}_i$ *is a finite set of* information sets $I \in \mathcal{I}$*.* $\mathcal{I}$ *is a partitioning over the nodes assigned to player* i $\{h \in \mathcal{H} : \mathcal{P}(h) = i\}$*. Every information set* $I \in \mathcal{I}$ *contains at least one node and each node belongs exactly to one information set. Nodes in an information set of a player are indistinguishable to the player. All nodes in* $h \in I_i \in \mathcal{I}$ *have the same set of actions* $\mathcal{A}_i(h)$*. Since each history* h *belongs to exactly one information set* I_i*, we overload notation* $\mathcal{A}_i(I_i)$ *to denote the set of actions defined for any node* $h \in I_i$*.*

We specify $ha = h' \in \mathcal{H}$ to be a node h' reached from node h by executing action $a \in \mathcal{A}_i(h)$. The game starts with empty history $h = \emptyset$. For history h we denote the sequence of player i as h_i. In this work we deal only with *perfect-recall* games, where players never forget the history of their own actions, which can be expressed as $\forall h, h' \in I \in \mathcal{I}_i : h_i = h'_i$. A game is termed to have *stochastic events* if in any history $h \in \mathcal{H}$ in the game the Chance player $\mathcal{P}(h) = c$ is assigned to play; otherwise, the game is without stochastic events. In *perfect-information* games all players are certain of their states (each information set contains exactly one game state). This is the case of games such as chess, go, tic-tac-toe, etc. If at least one information set contains more than one state, the game is called *imperfect-information*. An example of imperfect-information games is poker, battleship game, etc. A game is called zero-sum if $\forall z \in Z : u_i(z) = -u_{-i}(z)$; otherwise it is referred as to *general-sum game*.

Each normal-form game can be transformed into imperfect-information extensive-form game. Extensive-form game in Figure 2.2a is identical to the normal-form game in Table 2.1. Player 1 chooses a row $\{Up, Down\}$ which leads to an information set $I_1 \in \mathcal{I}_2$ for player 2. Player 2 cannot observe which action was played and chooses an action from $\{Left, Right\}$ to play in information set I_1.

The game in our example in Figure 2.2a is imperfect-information perfect-recall general-sum game. It consists of $|\mathcal{H}| = 7$ nodes and the actions that each player can play are

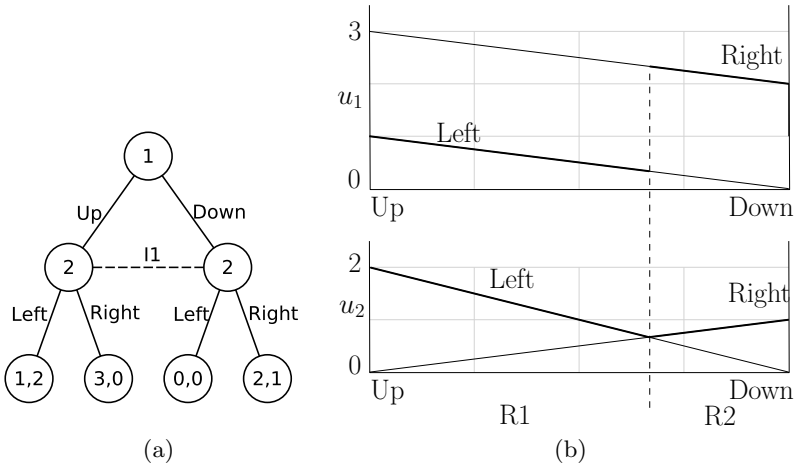


Figure 2.2: (a) A simple extensive-form game; (b) A simple analysis of information set I_1 from (a).

$\mathcal{A}_1 = \{Up, Down\}, \mathcal{A}_2 = \{Left, Right\}$. Player 1 has one information set (the root of the game $h = \emptyset$), player 2 has one information set $I_1 \in \mathcal{I}_2$ and there are $|\mathcal{Z}| = 4$ leaf nodes.

In extensive-form game, we define strategies similarly to normal-form games. A pure strategy $\pi_i \in \Pi_i$ of player i assigns for each $I \in \mathcal{I}_i$ an action $a \in \mathcal{A}_i(I)$, where Π_i is set of all pure strategies. In strategy profile $\pi = (\pi_i, \pi_{-i})$ the player i's expected utility $u_i(\pi)$ is computed as an expected utility of the terminal states reached by π. Note, that more than one terminal node can be reached in strategy with pure strategies because of the stochastic events of the Chance player. We define mixed strategies $\delta_i = \Delta(\Pi_i)$ of player i as a probability distribution over pure strategies. The player i's expected utility $u_i(\delta_i, \delta_{-i})$ in strategy profile (δ_i, δ_{-i}) is the expected payoff of terminal states reached by in this strategy profile.

Graphical Analysis An information set in EFG with few states and few outgoing actions can be analyzed graphically. In Figure 2.2b we analyze game in Figure 2.2a. In these plots player 1 chooses a point on x-axis, which mixes between actions Up Down. For pure strategies of player 2 (action Right or Left), we depict the expected utility u_1 and u_2 that each player obtains. If player 2 plays mixed strategy, the expected utilities of the players are mixed accordingly. For example, if player 1 plays uniform strategy (middle point of x-axis), player 2 receives maximal utility by playing action Left (higher on y-axis at the bottom plot). Using such analysis we can e.g. divide the space into the best response regions for player 2: in region R1 (resp. R2) player 2's best response is to play Left (resp. Right) against the player 1's mixed strategy. This analysis we be useful for analyzing solution concept, presented in this chapter.

2.3.3 Solution Concepts

A *solution concept* formally describes how a game will be played. Most of the solution concepts make use of *best response* function $BR_i(\delta_{-i})$ that returns a set of strategies of player i that yield the highest utility for i given that the opponent plays δ_{-i}. We additionally define ϵ-optimal version of the best-response function, that returns a set of strategies for player i, that can be improved by at most $100\epsilon\%$, if best-response is played instead. Formally,

$$\mathrm{BR}_i^{\epsilon}(\delta_{-i}) = \{\delta_i \in \Delta_i | \forall \delta_i' \in \Delta_i : u_i(\delta_i', \delta_{-i}) - u_i(\delta_i, \delta_{-i}) \leq \epsilon |u_i(\delta_i, \delta_{-i})|\}. \tag{2.6}$$

If $\delta_i \in \mathrm{BR}_i^{\epsilon}(\delta_{-i})$ then by playing any other strategy Δ_i the player i cannot improve his utility by more than $100\epsilon\%$ against strategy δ_{-i}. Setting $\epsilon = 0$ returns the optimal best-response strategy. When ϵ is omitted in $\mathrm{BR}_i^{\epsilon}()$, we refer to the optimal best response $\mathrm{BR}_i^{0}()$.

Nash Equilibrium

The most famous solution concept in game theory is the *Nash equilibrium* (NE), where all player's play ϵ-best response strategy against the opponent's strategy. Formally,

Definition 7 (ϵ-Nash equilibrium)**.** *Strategy profile $(\delta_1, \delta_2) \in \epsilon$-NE belongs to a set of ϵ-Nash equilibria, if in (δ_1, δ_2) both players play best response strategy against the opponents strategy. Formally, $(\delta_1, \delta_2) \in \epsilon\text{-}NE \iff \forall i \in \mathcal{N} : \delta_i \in BR_i^{\epsilon}(\delta_{-i})$.*

By setting $\epsilon = 0$ we obtain the exact Nash equilibrium, to which we refer as to 0-NE (or just NE). Strategy profile in ϵ-Nash equilibrium is stable since none of the player have incentive to deviate from that by more than ϵ. General-sum games may have multiple Nash equilibria and each equilibrium may result in different utilities to players. However, this is not the case for zero-sum games, where all NE have the same expected utility to both players. This utility value is called *value of the game*, denoted $\mathcal{V}^*$. A strategy profile in NE of a zero-sum game can be found in polynomial time in the size of a game (tree), e.g. by formulating it as a linear program and solving it. Computing NE strategy profile of a general-sum game belongs to harder complexity class PPAD.

Stackelberg Equilibrium

In games modeling security problems the researcher use *Stackelberg equilibrium* (SE) solution concepts, which is well motivated and justified theoretically and practically (in [Von Stackelberg, 2010, Tambe, 2011]). In SE one player called *leader* (with subscript l) commits to play a strategy δ_l and publicly announces it. The second player is the *follower* (with subscript f) who *knows* the leader's strategy δ_l and chooses a best-response strategy $\delta_f \in \mathrm{BR}_f(\delta_l)$ to play. This solution concept models a Kirchhoffs's principle, often assumed in security problems, which states that the system should be secure even if everything about the system is publicly known. In reality the attackers can learn the defense strategy either by purchasing identical algorithm used by the defender, or conducting a surveillance of the defender's behavior [Tambe, 2011].

To fully define Stackelberg equilibrium, we need to state which best-response action the follower chooses if more than one yield him maximal utility. A *strong* version of Stackelberg equilibrium assumes that the attacker chooses such best-response strategy, that also maximizes the leader's expected utility. This assumption ensures the existence of the equilibrium, since other variants, such as *weak* or *average* Stackelberg equilibrium, where the attacker chooses best-response with the lowest or an average expected utility for the leader, may not exist ([von Stengel and Zamir, 2010]).

Let $s(\delta_l) : \Delta_l \to \Pi_f$ be the attacker's response function. We define strong Stackelberg equilibrium (SSE) as follows:

Definition 8 (Strong Stackelberg equilibrium (SSE))**.** *A solution profile $(\delta_l, s(\delta_l))$ is a strong Stackelberg equilibrium if and only if:*

1. *The leader (defender) commits to a strategy that maximizes his utility:*

$$u_l(\delta_l, s(\delta_l)) \geq u_l(\delta'_l, s(\delta'_l)), \text{ for all } \delta'_l \in \Delta_l$$

2. *The follower (attacker) chooses best-response function s:*

$$u_f(\delta_l, s(\delta_l)) \geq u_f(\delta_l, \pi_f), \text{ for all } \delta_l \in \Delta_l, \pi_f \in \Pi_f$$

3. *The follower breaks ties in the favor of the leader:*

$$u_l(\delta_l, s(\delta_l)) \geq u_l(\delta_l, \pi'_f), \text{ for all } \delta_l \in \Delta_l, \pi'_f \in BR_f(\delta_l).$$

In words, in a Strong Stackelberg equilibrium the leader's strategy δ_l yields maximal expected utility to the leader assuming that the follower plays a best-response strategy $\delta_f \in BR_f(\delta_l)$ and breaks ties in favor of the leader.

Example 2.3.1. *Let us analyze game in Figure 2.2a (based on Table 2.1) for NE and SSE. One can easily see, that for any strategy of player 2, player 1 yields always higher expected utility for playing Up instead of Down. Therefore, action Down can be eliminated from consideration of being in NE and smaller version of the game can be analyzed. Here, player 1 must play Up, and player 2's best-response is to play Left. Indeed, strategy profile (Up, Left) forms a Nash equilibrium and yields utilities (1, 2) to players 1 and 2, resp.*

Let us look at strong Stackelberg equilibrium and let player 1 be the leader and player 2 be the follower. If the leader commits to play Down, the follower best-responds with Right which already yields higher expected utility of 2 to player 1 than he got in NE. However, the leader can further improve his expected utility by playing $\delta_l(Up) = 1/3$ and $\delta_l(Down) = 2/3$, represented by dashed line in Figure 2.2b. The follower obtains the same utility for either of his actions, he plays Right which maximizes the leader's utility. In this strategy profile, the leader yields utility $7/3 = 2.333$ (compare to expected utility 1 in NE) and the follower $1/3$.

This demonstration manifests the first-movers advantage in SSE. In SSE the leader yields at least as much expected utility as in any NE ([Conitzer and Korzhyk, 2011]). This proposition can be easily verified. Since in SSE the leader can commit to any strategy,

he can commit to play his strategy from any NE against which the follower best-responds in favor of the leader. This ensures him at least the utility of the best NE to the leader. Because in SSE the leader is not constrained to play a best-response against the follower's strategy, the leader can commit to a strategy with potentially even higher utility to him.

Beside the above mentioned disadvantage of NE, it additionally suffers with *equilibrium selection* problem: the players have no way of telling which equilibrium they should play if the equilibrium is not unique, and their mis-coordination may be arbitrarily bad for them. The Stackelberg equilibrium overcomes this problem by assuming that the follower knows the strategy of the leader and a plays best response. Therefore, in this thesis we focus on NE in zero-sum games and SSE in general-sum games.

Correlated Equilibrium

In Nash equilibrium, each player mixes his pure strategies independently. In some cases, players would benefit if they could correlate their actions. Correlated equilibrium, a generalization of the Nash equilibrium, solves exactly this problem. It introduces a device or mediator, that sends signals to the players with recommendation of what pure strategy they should play, allowing them to coordinate. The traffic lights at the road intersections play such a device. By signaling the color to the cars allows them coordinating their actions (*drive* or *stop*) and avoiding car accident. For normal-form games, *canonical representation*[1] of correlated equilibrium is as follows:

Definition 9 (Correlated equilibrium (CE))**.** *Let* $(\mathcal{N}, \mathcal{A}, u)$ *be a two-player NFG. Let* $p = \Delta(\mathcal{A}_1 \times \mathcal{A}_2)$ *be a probability distribution over the game outcomes and* $p_{(a_1,a_2)}$ *denote the probability of outcome* (a_1, a_2)*, where* $a_1 \in \mathcal{A}_1, a_2 \in \mathcal{A}_2$*. Probability distribution* p *forms a correlated equilibrium if it fulfills the* incentive constraints *that:*

$$(\forall a_1, a_1' \in \mathcal{A}_1) : \sum_{a_2 \in \mathcal{A}_2} p_{(a_1,a_2)} u_1(a_1, a_2) \geq \sum_{a_2 \in \mathcal{A}_2} p_{(a_1,a_2)} u_1(a_1', a_2) \tag{2.7}$$

$$(\forall a_2, a_2' \in \mathcal{A}_1) : \sum_{a_1 \in \mathcal{A}_1} p_{(a_1,a_2)} u_2(a_1, a_2) \geq \sum_{a_1 \in \mathcal{A}_1} p_{(a_1,a_2)} u_2(a_1, a_2') \tag{2.8}$$

The probability distribution p is known to both players. The mediator samples profile (a_1, a_2) with probability $p_{(a_1,a_2)}$ from p and recommends to each player i to play a_i without telling what recommendation was sent to the other player. The incentive constraints state, that for the players it is optimal to follow the recommendation.

Any Nash equilibrium can be interpreted as a Correlated equilibrium, where the mediator signals game outcomes with frequency of that in the Nash equilibrium. In [von Stengel and Zamir, 2010] the authors showed, that the leader's expected utility in SSE is at least as high as the utility in any correlated equilibrium in NFG.

[1] Canonical representation means, that the mediator signals the players' pure strategies that they should play. Other option is to use different signal domain than the players' pure strategies.

2.3.4 Smallest Undominated Set (SUS)

Many games can have a large set of actions for the players to choose from, which makes the computation of the desired solution concept very challenging or infeasible in these games. However, often not all actions must be considered in these games to provably conclude that the found strategy profile is the desired solution concept. Let us bring out other useful concepts that can significantly reduce the size of the game to speed-up the solution profile computation.

Consider two strategies, δ_i, δ_i' of player i. If strategy δ_i results in a lower utility to player i than strategy δ_i' *no matter what* strategy the opponent plays, then clearly δ_i can be removed from the consideration of being part of the Nash equilibrium. Here, we formally introduce the domination as follows: Let δ_i and δ_i' be two strategies of player i. Strategy δ_i *strictly dominates* δ_i' if $\forall \delta_{-i} \in \Delta_{-i} : u_i(\delta_i, \delta_{-i}) > u_i(\delta_i', \delta_{-i})$.

A strategy is *strictly dominant* for a player if it strictly dominates any other strategy for that player. And strategy δ_i is *strictly dominated* for a player if there is any other strategy of that player that strictly dominates δ_i. In this thesis by the dominations we always refer to the strict domination, if not stated otherwise.

Consider game shown in Figure 2.1 and corresponding analysis in Figure 2.2b. The player 1's action Down is dominated by action Up, since for any mixed strategy of player 2, action Up yields a higher utility than action Down. That is why in Example 2.3.1 we could eliminate action Down from the consideration of being in the NE and readily concluding that the solution profile (Up, Left) is in the Nash equilibrium.

If there is no strategy $\delta_i' \in \Delta_i$ that dominates strategy $\delta_i \in \Delta_i$, then δ_i is said to be *undominated* strategy. Given a set of opponent's pure strategies Π_{-i}, we further define player i's *smallest set of undominated strategies* (SUS) ([Schwartz, 1970]) as follows:

$$\kappa_i(\Pi_{-i}) = \{\pi_i \in \Pi_i | \exists \delta_{-i} \in \Delta(\Pi_{-i}) : \pi \in \mathrm{BR}_i(\delta_{-i})\} \tag{2.9}$$

In words, $\kappa_i(\Pi_{-i})$ contains all best response strategies of player i to *any* mixed strategy of player $-i$ taken from Π_{-i}. No strategy $\pi_i \in \Pi_i$ that $\pi_i \notin \kappa(\Pi_{-i})$ will be ever played as a best response against any mixed strategy $\Delta(\Pi_{-i})$, therefore, it can be omitted. Restricting the set of pure strategies of player i to $\kappa_i(\Pi_{-i})$ does not omit any Nash equilibrium since the player should never play dominated strategy. Algorithms such as Iterative Removal of Dominated Strategies iteratively uses this concept for both player's to reduce the game before engaging in the computation of a strategy profile in the Nash equilibrium.

However, in the strong Stackelberg equilibrium, only the follower's set of actions can be reduced to SUS. Since the leader does not play the best response, reducing his set of pure strategies to SUS could lead to computing a solution profile that is not the SSE. Consider an example in Table 2.1, where improperly eliminating action Down of player 1 will result is a different solution profile in SSE than the SSE solution profile of the full game.

2.3.5 Algorithms

In this section we present the algorithms for the presented concepts.

Computing SUS in EFG

Given set of the opponent's pure strategies Π_{-i}, we can reduce the player i's set of actions to SUS in NFG with the following approach. We test whether action $a \in \Pi_i$ belongs to $\kappa_i(\Pi_{-i})$ by verifying whether there exists a probability distribution δ_{-i} over Π_{-i}, such that a yields higher expected utility to player i than any other $a' \in \Pi_i$. Testing one action can be formulated as a linear feasibility program (LFP) (from [Benisch et al., 2010]) as follows:

$$\text{max: (no objective)} \tag{2.10a}$$

$$\text{s.t. :}(\forall a'_i \in \Pi_i \setminus \{a_i\}) : \sum_{a_{-i} \in \Pi_{-i}} u_i(a_i, a_{-i})\delta_{-i}(a_{-i}) \geq \sum_{a_{-i} \in \Pi_{-i}} u_i(a'_i, a_{-i})\delta_{-i}(a_{-i}) \tag{2.10b}$$

$$(\forall a_{-i} \in \Pi_{-i}) : 0 \leq \delta_{-i}(a_{-i}) \leq 1 \tag{2.10c}$$

$$\sum_{a_{-i} \in \Pi_{-i}} \delta_{-i}(a_{-i}) = 1 \tag{2.10d}$$

The variables are $\delta_{-i}(\cdot)$ and constants are $u_i(\cdot)$. Constraint (2.10b) ensures that in the found probability distribution $\delta_{-i}(a_{-i})$ action a_i dominates any actions a'_i. The remaining of the constraints ensure proper probability distribution of $\delta_{-i}(a_{-i})$. If LFP is not satisfied, then action a_i is never best response and it cannot be removed from Π_i.

Finding Nash Equilibrium in Normal-Form Games

This thesis focuses only on computing the zero-sum NE. Finding NE is known to be a polynomial problem in the size of the NFG assignment due to minimax theorem by von Neumann ([Von Neumann and Morgenstern, 2007]):

$$\max_{\delta_i \in \Delta_i} \min_{\delta_{-i} \in \Delta_{-i}} u_i(\delta_i, \delta_{-i}) = \min_{\delta_i \in \Delta_i} \max_{\delta_{-i} \in \Delta_{-i}} u_i(\delta_i, \delta_{-i}) = \mathcal{V}^*. \tag{2.11}$$

Intuitively, it states that maximizing the worst-case expected utility and minimizing the best-case expected utility results in equivalent expected utility to player i. To find player 2's strategy in Nash equilibrium, we can construct and solve linear program as follows:

$$\min \mathcal{V}_1^* \tag{2.12a}$$

$$\sum_{a_2 \in \Pi_2} \delta_2(a_2) u_1(a_1, a_2) \leq \mathcal{V}_1^* \quad \forall a_1 \in \Pi_1 \tag{2.12b}$$

$$\delta_2(a_2) \geq 0 \quad \forall a_2 \in \Pi_2 \tag{2.12c}$$

$$\sum_{a_2 \in \Pi_2} \delta_2(a_2) = 1 \tag{2.12d}$$

In this LP utility $u_1(\cdot)$ are constants and only $\delta_2(\cdot)$ and $\mathcal{V}_1^*$ are variables. Let us first examine constraint (2.12b), which states that for every pure strategy a_1, player 1 can obtain at most value $\mathcal{V}_1^*$ given player 2's mixed strategy δ_2. Those pure strategies, for which the expected utility is exactly $\mathcal{V}_1^*$ will be player 1's best response actions. The remaining of the constraints ensures, that $\delta_2(\cdot)$ is a proper probability distribution over strategies in Π_2. Constructing similar LP we can solve for NE of player 1.

Double Oracle (DO) algorithm ([McMahan et al., 2003]) can be used to solve very large zero-sum normal-form games, and Bošanský in ([Bošanský et al., 2014]) has extended the algorithm to zero-sum EFG. It uses similar approaches as column/constraint generation techniques to arrive to the solution without building the full linear program. Double oracle begins with a *restricted game*, which is a game consisting of a subset of actions for both players. It incrementally extends the game by adding certain actions of the players until a point is reached when a Nash equilibrium of the restricted game is also a Nash equilibrium of the original game.

In this thesis, we use a **Single Oracle** (SO) algorithm, a variant of a double oracle algorithm, where the action of only one player are restricted. The algorithm on NFG is presented in Algorithm 3, where G denotes the full NFG and G^m a restricted game in iteration m, which consists of $m+1$ actions of player i and all actions of player $-i$.

Algorithm 3 Single-Oracle Algorithm.

```
1: function SINGLE-ORACLE(G)
2:     G^0 ⊆ G
3:     m ← 1
4:     repeat
5:         (δ_i, δ_{-i}) ← NE(G^{m-1})
6:         π_{-i} ∈ BR_{-i}(δ_i, G)
7:         if u_{-i}(δ_i, π_{-i}) > u_{-i}(δ_i, δ_{-i}) then
8:             G^m ← G^{m-1} ∪ {π_{-i}}
9:         end if
10:        m ← m + 1
11:    until u_{-i}(δ_i, π_{-i}) > u_{-i}(δ_i, δ_{-i})
12:    return (δ_i, δ_{-i})
13: end function
```

In line 2 the algorithm initializes restricted game G^0 which contains all the actions of player i and only *one* (randomly chosen) action $a \in \mathcal{A}_{-i}$. In line 5 the algorithm computes m-th Nash equilibrium (δ_i, δ_{-i}) of restricted game G^{m-1} (e.g. by solving linear program 2.12a) and in line 6, it computes player $-i$'s best-response against strategy δ_i in full game G. Next, the algorithm simply tests, whether player $-i$ improved his expected utility by playing a best-response action from G compared to the expected utility in NE of G^m. If player $-i$ did not improve, then the found NE of G^m is also a NE of the full game. Otherwise, the best-response action is added into restricted game G^m of next iteration and the process is repeated. The algorithm always terminates, because every iteration at least one new action is added. Since there is a finite number of actions for player $-i$, the algorithm terminates in a finite number of iterations.

Finding Strong Stackelberg Equilibrium in Normal-Form Games

Computing strong Stackelberg equilibrium in general-sum normal-form games can be done in polynomial time in the size of the NFG. There are two commonly known approaches

of finding SSE, as described in [Conitzer and Korzhyk, 2011]. First approach, *Multiple LPs*, exploits SSE property that the follower's best-response is a pure strategy. For each of the follower's pure strategy $\pi_f \in \Pi_f$, the algorithm finds the leader's mixed strategy δ_l that maximizes the leader's utility under a constraint that π_f is a best-response to δ_l ($\pi_f \in \mathrm{BR}_f(\sigma_f)$), which can be solved with LP. The algorithm iterates over all $\pi_f \in \Pi_f$ and the leader chooses such δ_l, that had maximal expected utility to him.

The second algorithm, *Single LP* (SLP), introduced in [Conitzer and Korzhyk, 2011] avoids iterating over all $\pi_f \in \Pi_f$ by combining all the constraints into one LP. This linear program is $|\mathcal{A}_2|$ times larger, but allows a linear program solver to optimize the solving process more efficiently. The SLP formulation is as follows:

$$\max p_{(a_l,a_f)} u_l(a_l, a_f) \tag{2.13a}$$

$$\sum_{a_l \in \Pi_l} p_{(a_l,a_f)} u_f(a_l, a'_f) \leq \sum_{a_l \in \Pi_l} p_{(a_l,a_f)} u_f(a_l, a_f) \quad \forall a_f, a'_f \in \Pi_f \tag{2.13b}$$

$$\sum_{a_l \in \Pi_l, a_f \in \Pi_f} p_{(a_l,a_f)} = 1 \tag{2.13c}$$

$$p_{(a_l,a_f)} \geq 0 \quad \forall a_l \in \Pi_l, a_f \in \Pi_f \tag{2.13d}$$

In this linear program $p_{(\cdot)}$ are the only variables and $u_l(\cdot), u_f(\cdot)$ are constants. The objective (2.13a) of LP is to find such probability distribution $p_{(\cdot)}$ over the outcomes that the expected utility of a leader is maximized. The constraint (2.13b) states, that the follower should not be better off playing action a'_f instead of a_f. The constraint matrix of this linear program has blocks along the diagonal, one for each $a_f \in \Pi_f$. If LP solution has only single active block of nonzero variables, it corresponds to the leader's mixed strategy and followers best-response action a_f. The solver may find an optimal solution with more than one block where variables takes nonzero values. Nevertheless, from this solution we can easily compute the solution of the former form with the same expected utility to the leader and obtain SSE strategy profile. The proof of correctness of this approach and the transformation is shown in detail in Prop. 1 in [Conitzer and Korzhyk, 2011].

Computing Correlated Equilibrium in Normal-Form Games

The problem of computing correlated equilibrium of a normal form game can be formulated as a linear program, as follows:

$$\text{no objective} \tag{2.14a}$$

$$\text{s.t. :} (\forall a_1, a_1' \in \mathcal{A}_1) : \sum_{a_2 \in \mathcal{A}_2} p_{(a_1,a_2)} u_1(a_1, a_2) \geq \sum_{a_2 \in \mathcal{A}_2} p_{(a_1,a_2)} u_1(a_1', a_2) \tag{2.14b}$$

$$(\forall a_2, a_2' \in \mathcal{A}_1) : \sum_{a_1 \in \mathcal{A}_1} p_{(a_1,a_2)} u_2(a_1, a_2) \geq \sum_{a_1 \in \mathcal{A}_1} p_{(a_1,a_2)} u_2(a_1, a_2') \tag{2.14c}$$

$$\sum_{a_1 \in \mathcal{A}_1} \sum_{a_2 \in \mathcal{A}_2} p_{(a_1,a_2)} = 1 \tag{2.14d}$$

$$(\forall a_1 \in \mathcal{A}_1, a_2 \in \mathcal{A}_2) : p_{(a_1,a_2)} \geq 0 \tag{2.14e}$$

$$\tag{2.14f}$$

The only variables are $p_{(a_1,a_2)}$, which will compute the probability distribution over the game profiles. The constraints (2.14b,2.14c) incentivize the players to follow the mediator recommendation instead of playing any different action a_1' or a_2', resp.

Notice, that removing the player 1's incentive constraints (2.14b) from LP 2.14 and adding the objective function to maximize the player 1's expected utility, we will obtain LP 2.13. This demonstratively explains why in SSE the leader yields higher or equal utility than in any CCE. Removing any constraints cannot restrict the solution space and leader's utility is maximized.

Sequence-form Representation for Extensive-Form Games

The state-of-the-art algorithms for EFG use *sequence-form representation* to compactly represent the players' set of strategies ([Koller et al., 1996]), since converting EFG to NFG results in exponential large matrix game.

Definition 10 (Sequence)**.** *A* sequence $\sigma_i \in \Sigma_i$ *is an ordered list of actions taken by a single player* i *in a history* $h \in \mathcal{H}$*, where* Σ_i *is a set of all sequences of player* i*.*

The set of all possible sequences in a game for all players is $\Sigma = \Sigma_1 \times \Sigma_2$. A sequence $\sigma_i \in \Sigma_i$ can be extended by single action $a \in \mathcal{A}_i$ taken by player i, denoted as $\sigma_i a = \sigma_i'$. In games with perfect recall, all nodes in an information set I_i share the same sequence of actions σ_i for player i and we use $\text{seq}_i(I_i)$ to denote this sequence. We also use $\text{seq}_i(h)$ to denote the sequence of actions of player i leading to h and $\text{inf}_i(\sigma_i')$ to denote the information set in which the last action of sequence σ_i' is played (action a in this case). For an empty sequence, function $\text{inf}_i(\emptyset)$ is the information set of the root node.

Realization plan is the player's mixed strategy representation in sequence-form representation.

Definition 11 (Realization plan)**.** *A* realization plan *for player* i, $r_i : \Sigma_i \to [0,1]$ *satisfies the following constraints:*

$$r_i(\emptyset) = 1 \tag{2.15}$$

$$\sum_{a \in \mathcal{A}(I_i)} r_i(\sigma_i a) = r_i(\sigma_i) \quad \forall I_i \in \mathcal{I}_i, \sigma_i = seq_i(I_i) \tag{2.16}$$

$$r_i(\sigma_i) \geq 0 \quad \forall \sigma_i \in \Sigma_i \tag{2.17}$$

The constraints ensure the "waterfall" effect of the strategy. It states that the sum of the probabilities of the actions in an information set must equal to the probability of reaching that information set.

The *payoff* function $g_i : \Sigma_1 \times \Sigma_2 \to \mathbb{R}$ for player i extends the utility function to all nodes $\mathcal{H}$ in the game tree. Function g_i represents the expected utility of all nodes reachable by sequentially executing the actions specified in a pair of sequences σ:

$$g_i(\sigma_i, \sigma_{-i}) = \sum_{h \in \mathcal{Z}: \forall j \in \mathcal{N}: \sigma_j = \text{seq}_j(h)} u_i(h) \cdot \mathcal{C}(h).$$

If no leaf is reached by sequentially executing all action in sequences σ, then $g_i(\sigma) = 0$. This can happen in two cases: (i) either an inner node is reached after all sequences are executed in σ, for which the utility is not defined; (ii) or for a node $h \in \mathcal{H} \setminus \mathcal{Z}$ the strategy σ does not define any action $a \in \mathcal{A}(h)$ to be played.

Computing Nash Equilibrium in Zero-Sum EFG

We can use linear program to find a Nash equilibrium in zero-sum extensive-form game in a polynomial time in the size of the game tree using realization plans over the sequence-form representation. The linear program is as follows (from [Shoham and Leyton-Brown, 2008], p.135):

$$\max v(\text{inf}_{-i}(\emptyset)) \tag{2.18a}$$

$$\sum_{\sigma_i \in \Sigma_i} g_i(\sigma_i, \sigma_{-i} a) \cdot r_i(\sigma_i) + \sum_{I' \in \text{inf}_{-i}(\sigma_{-i} a)} v(I') \geq v(I) \qquad \forall I \in \mathcal{I}_{-i}, \forall a \in \mathcal{A}(I), \forall \sigma_{-i} \in \text{seq}_{-i}(I) \tag{2.18b}$$

$$r_i(\emptyset) = 1 \tag{2.18c}$$

$$\sum_{a \in A(I_i)} r_i(\sigma_i a) = r_i(\sigma_i) \qquad \forall I_i \in \mathcal{I}_i, \sigma_i = \text{seq}_i(I_i) \tag{2.18d}$$

$$r_i(\sigma_i) \leq 1 \qquad \forall \sigma_i \in \Sigma_i \tag{2.18e}$$

In this LP variables are $v(\cdot)$ and $r_i(\cdot)$, and constants are $g_i(\cdot)$. The solution contains realization plan for player i in variables r_i, and the expected utility for the information sets of player $-i$ in $v(I)$. In this LP, player i maximizes the expected utility of player i information sets by selecting the values for the variables of realization plan constrained by

(2.18c–2.18e). The realization plans are further constrained by (2.18b), so that the player $-i$ selects in each information set I such action that minimizes the expected utility in this information set $v(I)$. There is one constraint defined for each sequence for each sequence σ_{-i}.

Similarly as for NFG, we can use Double Oracle algorithm to find NE fast in EFG (described in [Bošanský et al., 2014]). The algorithm for EFG additionally has to deal with two complications that arise during the computation. First, strategy computed in the restricted game may not be a complete strategy in the original game, because it does not define behavior for information sets that are not in the restricted game. Second, It may not be possible to play every action from a sequence that is allowed in the restricted game, because playing a sequence depends on having a compatible sequence of actions for the opponent. However, the concept of the algorithms as described in ([Bošanský et al., 2014]) is the same.

Finding Strong Stackelberg Equilibrium in General-Sum EFG

Computing strong Stackelberg equilibria of imperfect information games with stochastic events is in general case NP-hard [Letchford and Conitzer, 2010]. First algorithms transformed EFG to NFG and used basic NFG algorithm to compute SSE ([Conitzer and Sandholm, 2006]). The state-of-the-art algorithm for solving this class of games uses mixed-integer linear programming (MILP) and avoids exponential representation by using the sequence-form representation of strategies [Bošanský and Čermák, 2015], which we use in this thesis. Recently in [Čermák et al., 2016a] the authors extended the algorithm to use iteratively approach to converge to SSE via *commitment to correlated equilibrium* solution concept. This approach allows reducing the number of binary variables in MILP and leads to faster computations.

In this thesis, we use the MILP approach described in ([Bošanský and Čermák, 2015]) as presented in Equations 2.19. In this MILP $p_{(\cdot)}, r_i(\cdot), v(\cdot)$ and $s_{(\cdot)}$ are variables. The MILP formulation exploits the fact, that the follower best-responds with a pure strategy, therefore, his realization plan can be represented as a binary variable $r_f \in \{0, 1\}$. Notice, that the LP is very similar to LP for NE in zero-sum EFG. The difference is basically in the fact, that the follower is allowed to play only pure strategies (2.19h). The sequences that the follower plays $r_f(\sigma_f) = 1$ force the slack variables $s_{\sigma_f} = 0$, which in turn forces these sequences to be best-responses in Constraint (2.19b). Additionally, the objective is to maximize the leader expected utility, which is achieved similarly as in LP 2.13 for computing SSE of NFG. It finds such probability distribution p_z over the terminal states $\mathcal{Z}$, that maximizes the leader's expected utility. Constraints (2.19f–2.19g) ensure, that probabilities of reaching

the terminal nodes p_z is consistent with realization plans of the players.

$$max \sum_{z \in \mathcal{Z}} p_z u_l(z) \mathcal{C}(z) \tag{2.19a}$$

$$\sum_{\sigma_l \in \Sigma_l} g_f(\sigma_l, \sigma_f a) \cdot r_l(\sigma_l) + \sum_{I' \in \inf_l(\sigma_l a)} v(I') + s_{\sigma_f} = v(I) \qquad \forall I \in \mathcal{I}_l, \forall a \in \mathcal{A}(I), \forall \sigma_l \in \text{seq}_l(I) \tag{2.19b}$$

$$r_i(\emptyset) = 1 \qquad \forall i \in \mathcal{N} \tag{2.19c}$$

$$r_i(\sigma_i) = \sum_{a \in \mathcal{A}(I_i)} r_i(\sigma_i a) \qquad \forall i \in \mathcal{N}, \forall I_i \in \mathcal{I}_i, \sigma_i = \text{seq}_i(I_i) \tag{2.19d}$$

$$0 \leq s_{\sigma_f} \leq (1 - r_f(\sigma_f)) \cdot M \qquad \forall \sigma_f \in \Sigma_f \tag{2.19e}$$

$$0 \leq p_z \leq r_i(\text{seq}_i(z)) \mathcal{C}(z) \qquad \forall i \in \mathcal{N}, z \in \mathcal{Z} \tag{2.19f}$$

$$\sum_{z \in \mathcal{Z}} p_z = 1 \tag{2.19g}$$

$$r_f(\sigma_f) \in \{0, 1\} \qquad \forall \sigma_f \in \Sigma_f \tag{2.19h}$$

$$r_i(\sigma_i) \leq 1 \qquad \forall \sigma_i \in \Sigma_i \tag{2.19i}$$

Chapter 3

Security Configuration Problem

The network administrators face a challenging decision-making problem of finding the best system security configuration for a set of systems that they secure. The complexity of the problem stems from the combinatorial expansion of the available configurations actions to the network administrator, and the reasoning about the opponents reaction to defense measures.

System mis-configuration may results in vulnerable systems that adversaries can exploit to harm the users. To address this problem, a discipline security configuration management was established that aims to devise and deploy policies that carry out organization requirements to oppose the attackers ([Couch, 2008, Bellovin and Bush, 2009]). Its major challenge is scaling for large system, since configuring 1,000 or 10,000 of devices securely and consistently is significantly more difficult from configuring each machine independently. Configuring one machine may have a direct or indirect influence on the other machines. E.g., increasing the protection of one machine can result in the adversary attacking different machine instead.

Security organization also address the problem by designing best-practice standards such as NIST 800-41-rev1, a guideline on firewall security configuration. Automatic security configuration tools exists that usually verify the security configuration checklist (e.g., [Peterside et al., 2015] for Cisco IPsec VPN) or best practices ([Fitzgerald et al., 2013] for the smart phones). However, these tools are design for specific devices and do not consider the attacker's strategic reasoning and counteractions to the configuration actions.

In this chapter, we introduce a specific game-theoretic model that captures the interaction between the defender, who configures multiple networks and the attacker, who attacks them. Our game explicitly models the attacker's knowledge about the possible real systems and the observed information about the configured systems during his attack. Formalizing the attacker's reasoning about the systems allows us developing defense strategies against the attackers that act optimally given the available information to them. The model captures essentials of information reasoning in configuration problem and in Chapters 4 and 5 is extended to specific security problems.

3.1 Game-Theoretic Model

Consider a motivating problem with a game-tree depicted in Figure 3.1 as follows. The network administrator aims to protect three of his systems s_1, s_2 and s_3. For each system the defender can choose either to increase its security (action c_s), or to modify the system to mimic a different system (actions c_m). The defender can mimic system s_1 only as s_2 (action c_{m12}), system s_2 as system s_1 or s_2 (actions $c_{m_2}1, c_{m_2}3$, resp.), and system s_3 as s_2 (action c_{m32}). Here, the defender faces dilemma: playing action c_s in each system will increase their security. However, the attackers can observe the configured state exactly and perform tailored attack for that system. On the other hand, if the defender mimics some (or all) systems as a different system, the attacker is uncertain of the attacked system, therefore, does not know what attack to perform. However, mimicking systems leave them more vulnerable. Each of these extreme cases have some advantages and disadvantages and the defender must weigh benefits and drawbacks of each approach. We use this running example to present our general extensive-form general-sum game theoretic model.

3.1.1 Chance Actions

Skilled attackers have good knowledge about the possible systems in the real-world that they may encounter during an attack. In the *first stage* of the game, we model the attacker's beliefs about possible systems by a *Chance* (or Nature) player who makes an initial random move. The Chance player chooses a system $s \in \mathcal{A}_c$ with probability $\mathcal{C}(s)$ which corresponds to the attacker's belief of the likelihood that he will attack system s. In Figure 3.1 the attacker knows that there are three systems and he may encounter any of them with an equal probability of 1/3.

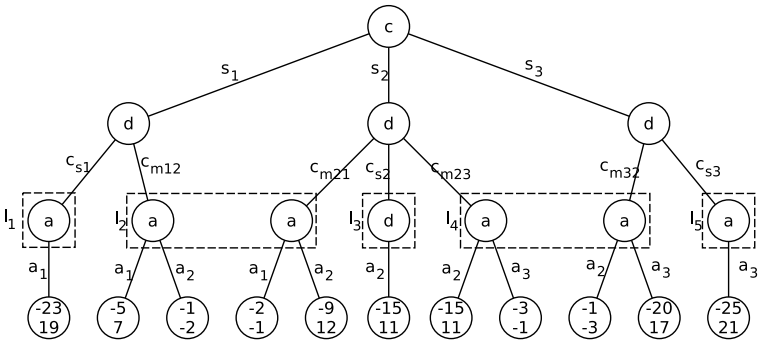


Figure 3.1: A simple game tree with three systems s_1, s_2 and s_3 chosen by the Chance each with probability 1/3. The defender configures each system from 2 possible configurations and the attacker chooses among six different attacks in total.

3.1.2 Defender's Actions

In the *second stage*, the defender chooses a configuration action $c \in \mathcal{A}_d(s)$ to deploy in system $s \in \mathcal{A}_c$. Configuration c applied on system s results in configured system sc. Here, we present an abstract configuration action. In a specific problem it can represent as simple action as selecting a threshold parameter, or as complex action as fully configuring a system (e.g. honeypots). In our running example, the defender choose either to secure a system (action c_s) or to mimic a system as a different system (actions c_m).

3.1.3 Attacker's Information and Actions

After the defender acts, the attacker observes configured system sc (not the configuration action per se). Some configured systems may look alike to the attacker and since the attacker cannot tell them apart, they belong to the same information set $I \in \mathcal{I}_a$. In the *third stage* of the game, the attacker chooses an optimal attack action $a \in \mathcal{A}_a(I)$ for every information set $I \in \mathcal{I}_a$. Although a particular attacker in reality attacks only a single system (depending on the choice of Chance), to fully characterize the attacker's behavior we need to describe their behavior in *every* possible scenario that may occur to the attacker. The attackers' actions can be as simple as verifying if service on port is alive (e.g., pinging) or as complex as stochastic decision policy of sequential exploits.

In Figure 3.1, the attacker faces five information sets I_1, I_2, I_3, I_4 and I_5, where he makes his first decision about an attack action. E.g., information sets I_2 consists of two states and the attacker cannot be certain whether he faces a system s_1c_{m12} or s_2c_{m21}. The attacker must choose the same attack action for all the state in an information set since they are indistinguishable to him. But different information sets may have different attack actions to choose from.

3.1.4 Player Utilities

For terminal state with history $sca \in \mathcal{Z}$ we denote the player payoffs as:

$$u_d(sca) = -R_d(sca) - C_d(sc) \tag{3.1a}$$

$$u_a(sca) = R_a(sca) - C_a(sca) \tag{3.1b}$$

where:

- $R_d(sca) \geq 0$ is the defender's loss ($-R_d(sca)$ is his reward) in system s for configuring it with c and having it attacked with a. This component may capture a company's financial costs and other ramifications using utilitarian valuation.
- $R_a(sca)$ is the attacker's reward for attacking system sc with attack a, e.g., reward for selling the stolen data on gray market.
- $C_d(sc)$ is the defender's cost for deploying security configuration c on system a.
- $C_a(sca)$ is the attacker's cost for executing attack action a on configured system sc. It may include purchasing and exploiting a zero-day vulnerability, cost for being detected, etc.

Additionally, the attacker typically gains less than or equal to the value that the defender looses ($R_a(sca) \leq R_d(sca)$). Otherwise, the defender could gain by giving up the systems on his own. In some cases, we even assume it as a zero-sum component ($R_d(sca) = R_a(sca)$).

3.1.5 Game Properties and Assumptions

In this section is devoted to discussions about the game assumptions and properties.

Chance Probabilities are Common Knowledge

Our game assumes that the probabilities of Chance player actions are known to both players. The administrators indeed have knowledge about the systems that they secure. The attackers may not have as precise knowledge about the systems as the defender, but they can often estimate it based on other information, such as from systems in similar companies, from the published work of the security researchers, etc. Moreover, the attacker's small belief fluctuations are not as important for our model as their average belief over many attackers being close to the correct probabilities of the Chance actions.

System Types

The scalability of the algorithms is one of the major problems in algorithmic game theory. In our game, systems that manifest similar and indistinguishable properties to the attacker can be grouped together into a *system type*. The probability of a system type equals to the sum of probabilities of individual systems of that type. Hereafter, we refer to system s_i as a type of a system. In this case, the complexity of the model scales with the number of system types rather than the number of the systems itself, which can allow significant model reduction without any loss of solution optimality.

In our running example, each system s_1, s_2 and s_3 can represent hundreds or thousands of real systems. If the defender plays mixed strategy in these systems, he randomly samples a pure strategy for each system of a given system type based on the strategy. We use such system aggregation into system types in both of our domain-specific problems and refer to systems as system types.

Attacker's Sequential Actions

In our running example in Figure 3.1 the attacker performs single attack action after which the game ends. In reality the attack models are sequential. For instance, the attacker may perform series of probing and based on their results the attacker decides what attack action to perform next. The attacker's sequential decision can be straightforwardly used in our game model. However, sequential representations significantly increase the SSE computation, since each of the attacker's action introduces one binary variable to the MILP. These MILPs are often infeasible to solve and other techniques must be used. For this reason, in Chapter 4 we represent each sequence of actions as a single action, *but* use SUS to significantly reduce the final set of actions.

Information Manipulation

A key feature of our game model is that the defender can influence the attacker's likelihood of observing the configured systems in the information sets. The extent to which the defender can influence the attacker's observations depends also on the Chance player probabilities. In our example in Figure 3.1, the defender could choose to secure the systems, which does not introduce any uncertainty for the attacker; or to mimic one system as a different system and increase the attacker uncertainty about the state of a system, that the attacker attacks. In the case the attacker does not know the real system exactly (more than one systems are likely in an information set), the attacker chooses such action, that yields him the highest expected reward *given* the probability distribution of the systems in the information sets. Therefore, increasing the attacker's uncertainty in the information sets may push the attacker to perform broad and weaker attacks.

Defender Protects All Systems

Our model currently assumes that single defender configures *all* the system after the Chance plays, and that the defender maximizes the expected reward of all the systems. A reasonable question is *what if the defender does not control all the systems after the Chance player?* In reality, the network administrator may control only part of the systems (e.g., only those that are within the same company). In this section, we discuss the complexity that the relaxation of the assumption arises and the explain why we restrict our research to the cases with the above-mentioned assumption.

The systems, that are out of the defender's control, could be either (i) set statically prior to the defender's strategy computation without ability to adapt to the defender's strategies, or (ii) controlled by other self-interested rational decision-makers that react on the defender's strategy. To model the first case, the systems that are not under the defender's control can be replaced with Chance nodes with a static distribution of the actions corresponding to a static strategy that is played in those systems. Additionally, the defender's payoffs in the leaf nodes rooting from the new Chance nodes must be set to zero, since the defender has no motivation to protect them (while the attacker's payoffs are kept unchanged). However, assuming non-reactive defenders is myopic since owners of these systems will experience losses and eventually *will* decide to react. This corresponds to the second case discussed below and thus, the first case we omit in our analysis.

The second case assumes that other systems could be under control of different decision-makers, which then results in game with more than two players. In [Smith et al., 2014, Lou et al., 2015, Lou and Vorobeychik, 2015] authors attempted to extend strong Stackelberg equilibrium for games with multiple leaders (defenders). The main problem with multiple leaders is that each leader assumes that the follower (the attacker) will break ties in his favor. This assumptions obviously cannot be always met, because there may be the cases that the attacker cannot play in favor of *both* (or more) leaders at the same time. Since the attacker must choose an attack, for some leader the assumption will be violated, which means that this solution concept may not always exist.

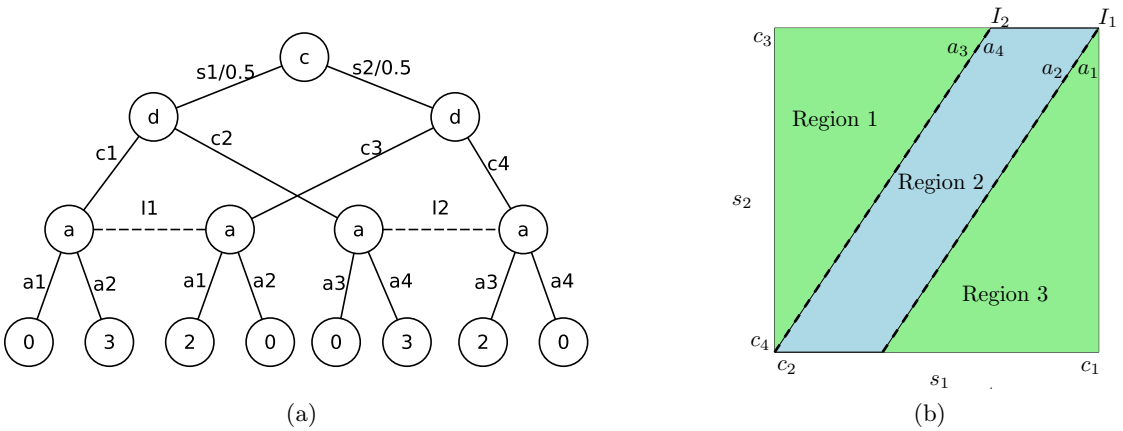
In ([Smith et al., 2014]) authors introduced an *Average-case Stackelberg Equilibrium*, where the attacker breaks ties at random. This assumes that all defenders simultaneously commit, each to their mixed strategy, and the attacker subsequently chooses a best-response uniformly at random. However, this solution profile does not necessarily always exists either. In the paper, the authors overcome the issue by computing ϵ-Nash Equilibrium, with the smallest ϵ such that the equilibrium exists.

In [Laszka et al., 2016b] the authors define alternative solution concept *Stackelberg Multi-Defender Equilibrium*. It assumes that all defenders play a best response against the others, assuming that the attacker will always play a best-response strategy. This solution concept assumes that the defenders play NE among each other and the attacker best responds to defenders' strategies. In their specific game, if such strategy profile exists all the players play pure strategies. The attacker does not have to choose one out of multiple best responses, which avoids the typical problem *which* best-response should he chose. However, even for their game such strategy profile does not necessarily exist. Since our payoffs do not allow concluding that all players play pure strategies, this solution concept is not applicable to our game either. Let us demonstrate a simple example, where *no stable solution* exists to see the difficulties of the multi-defender games.

Example 3.1.1. *Consider a game tree depicted in Figure 3.2a with two systems s_1, s_2. Each has probability 1/2 of reaching due to the Chance player and is controlled by a different defender. Let defender of s_1 (resp s_2) denote with d_1 (resp. d_2). Both defenders can configure their system to be either in information set I_1 or I_2, or mixing between the two. In each information set the attacker chooses which system to attack: actions a_2, a_4 attack system s_1 and actions a_1, a_3 attack system s_2. Terminal state present the attacker's payoffs and let the game be zero-sum.*

Both defenders prefer that the attacker attacks the other defender in the information sets. However, in this example at least one defender can always change his strategy to make the attacker to attack the other defender in at least one of the information sets. We can graphically demonstrate the defender's cycle reactions in Figure 3.2b. It shows the defenders' full strategy spaces. Defender d_1 chooses action c_1 (right), c_2 (left), or mixes the two. Defender d_2 chooses action c_3 (top), c_4 (bottom), or mixes the two. Each point in the square represents the defenders' strategies and the attacker's beliefs in the information sets. Diagonal lines represent the attacker's belief when he is indifferent whether to perform attack a_1 or a_2 in I_1 (right line). Similarly diagonal I_2 (left line) separates the beliefs where the attacker best-responds with a_3 and a_4. From this, we can define three regions, each with different strategy of the attacker. In Region 2 the attacker attacks s_1 in both information sets (actions a_2 and a_4), and in Region 1 and 3 the attacker attacks s_2 at least in one information set. Now we are ready to see the defenders' cycle in reasoning. The defender d_1 can always play a strategy to result in a point outside of Region 1, while the defender d_2 can always adjust his strategy to a result within *Region 2. Therefore, there is* no *stable solution profile where both defenders would not be incentivized to change their strategies.*

The reason why game in Example 3.1.1 has no stable solution is that in SSE the attacker plays pure strategy. If we allow the attacker to play a mixed strategy, then there exists a stable solution profile, namely Nash equilibrium, where no player has an incentive to deviate



(a) (b)

Figure 3.2: (a) Game, where Strong Stackelberg equilibrium does not exist if two systems (s_1 and s_2) are configured with two different rational decision-makes. (b) Attacker's best-response regions.

from it unilaterally. However, there is no particular reason *why* should the attacker play this specific mixed best-response strategy in NE. Moreover, in Nash equilibrium, the defender's give up the first-mover's advantage and may obtain lower utility that they possibly could.

As a result of this discussion, for the defender it is better to cooperate and to jointly commit to a strategy that maximizes their overall expected utility instead of maximizing their individual expected utilities. They could, e.g., use an *insurance mechanism* to mitigate the loss of the frequently attacked defender by paying him a premium. The premium should be high enough to incentivize this defender to play the strategy profile. We believe that using insurance mechanism the stable solution profile will exist. However, this requires further analysis which we leave for future work.

Due to the above-mentioned reasons, we restrict our research to a case, where a single defender (e.g. an insurance company) configures all the systems with a single objective to maximize his expected utility.

3.2 Related Models

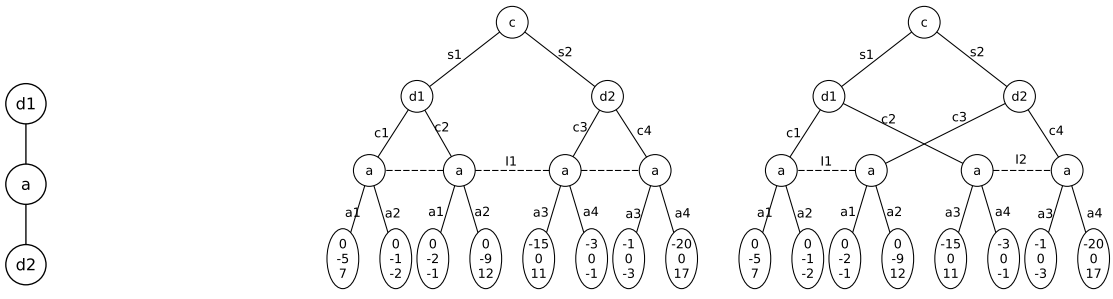
Game theory provides library of different models, e.g., Bayesian games, congestion games, graphical games, that are suitable for modeling and solving different classes of the problems. For these classes often more efficient algorithms can be used and/or more compact representation of the problems exist, that would otherwise result in exponentially large extensive-form representation game trees. In this section, we list two game theoretic models that are closely related to our game, but unfortunately, none of them is found to be suitable for our problem. Therefore, we use general EFG to model our problem and EFG general algorithms to solve it.

Bayesian games model the players' incomplete information about their opponents, such as their payoffs, using Harsanyi's transformation ([Harsanyi, 1967]). The transformation assumes, that players know the probability distribution of all possibilities of the payoffs, that the opponents might have, and modifies the original game by introducing an initial Chance player that chooses the player's type with the associated payoffs of that player.

Notice the similarity to our game, where in our case the Chance player selects the system types. However, our game is more general than a typical Bayesian model. In our game, the defender may have different set of actions for each system type, since the administrators of different systems might have different applicable configuration actions. Bayesian games typically assume, that for each type the player has an identical set of actions to choose from.

Graphical games is a compact representation that models limited interactions between two or more players in a game. This representation exploits the fact that the payoffs of one player can depend on a subset of the players' action choices instead of all the players. Graphical games represent these relations by a graph: each node represents a player, and each edge connects two nodes only if their payoff functions are depended on the actions choices of the players represented by the two nodes. Stated differently, if there is no edge between nodes i and j, then player i's choice of action does not influence the payoff of player j and vice versa. In Figure 3.3(left) is a graphical game with three players, where payoffs of the players represented with ds depend only on the player represented with node a. Graphical games have been used to show e.g., that n-player two-action games with tree-structure graphical games can be solved in polynomial time ([Kearns et al., 2001]),

In the case that our game is modeled with multiple defenders (that now we model as a single defender), the deviations of one defender have no impact on the utility of the other defender. Therefore, each defender plays a direct game *only* with the attacker. This could be represented as a graphical game with a star-like shape: the attacker is the center node, and each defender is the leaf node directly connected to the center node. However, as far as we know, the graphical models have no way of modeling the information sets that the



(a) (b) (c)

Figure 3.3: Graphical game in (a) corresponds to extensive-form game in (b). In (b) the order of the payoffs is d_1, d_2, a. In (c) is an example of our game models that cannot be modeled with graphical games.

defender plays into to deceive the attacker. E.g., consider a graphical game in Figure 3.3a that models the interaction between one attacker and two defenders, where each player has two actions. This game corresponds to the EFG depicted in Figure 3.3b. The attacker does not know which action each defender has chosen. However, he knows exactly what original system the attacker attacks. In contrast, our game is more general, as is depicted in Figure 3.3c, where the attacker is uncertain about the original system that the graphical games are incapable of modeling.

As far as we know, the general-sum imperfect-information EFG with stochastic events is the most specific class of games that is suitable for our problems. We use this class to model our problems and algorithms of this class to compute the desired solution concepts.

Chapter 4

Honeypot Selection Problem

The increasing complexity of securing modern computer networks makes decision support systems an important tool for administrators. A challenge many existing tools fail to address is that attackers react strategically to new security measures, adapting their behavior in response. Game theory provides a methodology for making decisions that take into account these reactions, rather than assuming static attackers. In this chapter, we present how game theory can be used to inform one type of security decision: how to optimally place honeypots in a network. The network administrator allocates honeypots, a fake services, into the networks to detect the malicious activity. While the attacker attacks the networks according to compact representation of all known vulnerabilities for the networks, called *attack graphs*. This problem can be formulated as a three-staged game, presented in Chapter 3. In this chapter we demonstrate this approach and present models, exact and heuristic algorithms that compute SSE.

4.1 Introduction

We demonstrate our methodology for a case where a network security administrator uses *honeypots* (fake hosts or services added to a computer network) to harden a network against possible attacks ([Qassrawi and Hongli, 2010]). Legitimate users do not interact with honeypots, so the honeypots act as decoys that distract the attackers from real hosts and waste their resources. Honeypots can also send intrusion detection alerts to the administrator, and/or gather detailed information about the attacker's activity ([Provos, 2004, Grimes, 2005]). Believable honeypots, are costly to set up and maintain, and a well-informed attacker may anticipate the use of honeypots and try to avoid them. To capture the strategic decision of both attackers and defenders, we model the problem as system configuration problem, where by configuration we mean deciding which services or host to deploy as honeypots.

In this chapter, we heavily use a compact representation of strategies for attacking computer networks called *logic-based attack graphs*. They can be automatically generated based on known vulnerability databases ([Ingols et al., 2006, Ou et al., 2006]) and they are used in network security to identify the minimal subset of vulnerabilities/sensors to be

fixed/placed to prevent all known attacks ([Sheyner et al., 2002, Noel and Jajodia, 2008]), or to calculate security risk measures, e.g., the probability of a successful attack ([Noel et al., 2010, Homer et al., 2013]). We use attack graphs to represent an attack plan library, from which a rational attacker will choose an optimal contingency plan. In real attack scenarios attackers do not know everything about the target network, and must make decisions based on imperfect information. We model this uncertainty that attackers have by introducing attack graph games with imperfect information. Here, the main idea is that attackers have beliefs about what realistic networks look like, and use this to distinguish between hosts/services that are likely to be real and those that are likely to be honeypots. However, generalizing our models in this way dramatically increases the computational challenge of solving the models, since attackers may believe that a vast number of networks are possible. We present the exact, error-bounded and a variety of heuristic algorithms for computing Stackelberg equilibria for attack graph games with imperfect information.

The structure of this chapter is as follows. In Section 4.2, we present the related work; in Section 4.3, we introduce attack graphs, which are used to compute the attacker's attack policies. In Section 4.4, we present our game-theoretic model for honeypot selection problem and concepts of the heuristic approaches and in Section 4.6 we describe the algorithms we use to computing strong Stackelberg equilibrium. In Section 4.7, we analyze our algorithms, including the scalability and quality of the strategies that they found as well as network security related questions using our models.

4.2 Related Work

4.2.1 Attack Graphs

In the context of network security, attack graphs identify all known attack paths the attacker could follow to exploit known vulnerabilities to compromise a specific network. *State-based attack graphs* ([Sheyner et al., 2002]) consist of network states, which scales exponentially with the increasing number of logical statements about the network. Therefore, a more efficient *logic-based attack graphs* representation was introduced, consisting of *fact nodes*, that can be either true or false, and the *action nodes*, that can modify the facts. There are number of tools that generate such attack graphs, such as *topological analysis of network attack vulnerability* (TVA) from [Jajodia et al., 2005, Noel et al., 2010], *a network security planning architecture* (NETSPA) from [Lippmann et al., 2002] and *multi-host, multistage, vulnerability analysis* (MulVAL) from [Ou et al., 2005].

Attack graphs are used to check the existence of sequences of actions that could lead to compromising the network ([Ritchey and Ammann, 2000]), a minimal set of vulnerabilities that must be patched to guarantee that the intruder cannot achieve their goal ([Barik et al., 2016, Sheyner et al., 2002, Jha et al., 2002]), or perform network risk assessment ([Cheng et al., 2014, Nandi et al., 2016]). In papers [Buldas and Stepanenko, 2012, Ou and Singhal, 2012, Wang et al., 2008a] and [Hu et al., 2017] the authors analyze attack graphs with probabilistic outcomes of the actions and introduce various quantitative metrics for networks risk assessment. These probabilities can be estimated from historical data, red team exercises or Common Vulnerability Scoring System (CVSS) presented in [Schiffman

et al., 2004] that compute the overall score of the vulnerability based on vulnerability attributes. We use probabilistic actions in attack graphs as well.

Attack Planning

The attack graphs with deterministic actions are also used in combination with the classical planning, where researchers find a sequence of actions that lead to a specific goal ([Boddy et al., 2005]) or plans that meet certain constraints ([Polatidis et al., 2017]). These plans can be used to predict the attacker's behavior or for penetration testing to automate generating of the attack scenarios to ease the task of searching the likely exploitable attack paths ([Obes et al., 2013]). Recently, in [Speicher et al., 2018] the authors introduce a new discipline Stackelberg planning. It consists of two decision-makers, where leader plans a sequence of actions from initial state to a state s, and then the follower plans from s to a goal state. Both players use STRIPS planning formalism, which results in large state space; on the other hand, the full game is deterministic and perfect-information. In [Sarraute et al., 2012] the authors introduced a POMDP model that includes the attacker's uncertainty about the network configuration, while actions are still deterministic.

Planning with probabilistic actions is more complex since plans are not *linear* anymore but *contingent*, specifying the attacker course of actions in the case the previous attempt of action failed. Algorithms that find optimal contingency plans do not scale well ([Durkota and Lisý, 2014]) and heuristic algorithm have been presented ([Sarraute et al., 2012]). In [Bi et al., 2016, JIA and Zhi-Chang, 2010, Borbor et al., 2017] the researchers focus on finding the most probable attack paths to identify areas to improve the security, or to replan if previous action should fail [Krautsevich et al., 2012].

Attack Graph Games

The main motivation to use game-theory in combination with the attack graphs is to model the dynamics of the attacker's complex reaction to the defender's defense measures. Static defense measures, such as presented in [Barik et al., 2016, Sheyner et al., 2002, Jha et al., 2002] can result in more expensive security measures than are necessary to demotivate the attacker from attacking a network. Modeling the attacker's possible reactions to the defense strategies can lead to more efficient and robust defense strategies.

In [Bistarelli et al., 2006a] the authors published the first attack tree game, where the attacker chooses a path from the root to a leaf (goal) according to the attack tree, and the defender chooses a defense measure. A game-theoretic model similar to ours is presented in [Letchford and Vorobeychik, 2013], where the attacker chooses an attack based on the attack graph that maximizes his expected utility and the defender decides which vulnerabilities to fix in the attack graph to maximize his own expected utility. In contrast to this work, we do not assume deterministic actions outcomes; additionally, we compute strategy for multiple networks at once, which allows exploiting the attacker's uncertainty about the attacked networks.

4.2.2 Honeypots

Decoy-based deception mechanism, e.g. *honeypot*, is a network component that serves to deceive the attacker into believing that it is a valuable target ([Almeshekah and Spafford, 2016]). In [Spitzner, 2003b] the researchers use honeypots to detect and analyze attackers' behavior and the tools they use. For example, the Honeynet Project in [Spitzner, 2003a] provides software and literature describing the knowledge acquired about the attackers. Companies use honeypots as intrusion detection systems (IDS), e.g., a hardware product called *Canary* produced by company Thinkst is a simple honeypot that can emulate various operating systems.

One of the basic deception game was introduced in [Carroll and Grosu, 2011], where the authors use signaling game to analytically compute a strategy, which prescribes when a production host should pretend to be a honeypot and when a honeypot should pretend to be a production host. In [Hayatle et al., 2012] the authors introduce the game, where a botmaster (attacker) tries to discriminate the honeypot services from the production services by executing testing and attacking actions. On the other hand, the honeypots decide what to reply to the attacker to be as believable real service as possible, and whether to execute or not the attacker's potentially dangerous command. In [Albanese et al., 2016, Jajodia et al., 2017] the authors investigate the responses the defender should reply to the attacker to maximize the defender's utility in more detail.

In this work, we investigate where in the network to deploy the honeypots and what services they should provide rather than what answer they should respond to the attackers. In [Garg and Grosu, 2007, Píbil et al., 2012] the authors model game-theoretically the situation without attack graphs, where the attacker has a direct visibility of the hosts and decides which hosts to probe and then to attack. Our work is distinguished from the above in several aspects. We consider the attacker's contingency plans based on the attack graphs generated by MulVAL, which fully characterize the attacker's behavior and allows the defender to strategize against all possible scenarios. Parts of our work were published in [Durkota and Lisý, 2014, Durkota et al., 2015b], where the attackers are assumed to have perfect information about a network; in [Durkota et al., 2015a] we introduce heuristic algorithms and compared them to a upper bound on the defender's utility in Stackelberg equilibrium; in [Durkota et al., 2016] we compared game-theoretic strategies to human strategies and analyzed when the game-theory is superior to humans and otherwise.

4.3 Attack Graph

An attack graph (AG) compactly represents all known sequences of attack actions for a specific network.

Definition 12 (Attack Graph)**.** *For network s we define an attack graph $AG(s)$ as a tuple $\langle T, F, A, p_a, c_a, \tau_a, r_f \rangle$, where:*

- *T is the set of host types in network s.*
- *F is the set of facts about network s that can be either true or false (e.g., whether the database has a specific exploit or not).*

- *A is the set of atomic actions available to the attacker. Each action $a \in A$ has a set of* preconditions *$pre(a) \subseteq F$ that must be true before the action a can be performed, and the set of* effects *$eff(a) \subseteq F$ that becomes true if action a is successfully performed.*
- *$p : A \to [0, 1]$ is the success probability of action $a \in A$. It represents the likelihood, that the attacker successfully executes the exploits and achieves the action's effects. We use p_a to denote this probability. The probabilities can depend on the attacker's expertise or even on the behavior of the user at the attacked host.*
- *$c : A \to \mathbb{R}^+$ is the attacker's cost for performing action a regardless of whether the action succeeded or not. The cost represents the time and resources that the attacker spends to perform the action.*
- *$\tau : A \to 2^T$ is the set of hosts that action a interacts with. We use τ_a to denote the set of hosts. We assume, that the first time the attacker interacts with a type $t \in T$, a specific host of that type is selected with a uniform probability. Future actions with the same host type interact with the same host. There is no reason to interact with a different host of the same type because (1) rewards, preconditions and effects are defined based on the types, so there is no additional benefit, and (2) interacting with another host increases the probability of interacting with a honeypot and ending the game.*
- *$r : F \to \mathbb{R}^+$ is the attacker's reward for making fact node f true (e.g., an attacker gains reward by gaining access to a particular host or compromising a specific service). We use r_f to denote the reward for reaching fact node f.*

We overload notation and use $eff(f) = \{a \in A : f \in pre(a)\}$ and $pre(f) = \{a \in A : f \in eff(a)\}$ to denote the fact's effects and preconditions, respectively. We use the common monotonicity assumption from [Ammann et al., 2002, Ou et al., 2006, Noel and Jajodia, 2004] that once a fact becomes true during an attack, it can never become false again as an effect of any action.

Example 4.3.1. *An example attack graph is depicted in Figure 4.1. Diamonds (resp. rectangles) are fact nodes that are initially* false *(resp.* true*). Rounded rectangles are actions, which are labeled with a triplet (p_a, c_a, τ_a). The attacker attacks in a top-down manner and his goal is to achieve root privileges in database host for which he gets reward +1000. At the beginning, the attacker can perform actions 6:Exploit Vulnerability, 7:Exploit Vulnerability or 8:execCode(PC, User), since their preconditions are initially true. The action 6:Exploit Vulnerability has precondition {9:vulExists(WebServer, 'CVE-2012-0572', MySql)}, and effect {5: netAccess(Database, '139', TCP)}. If this action is performed, the attacker immediately pays 5, and interacts with a WebServer host type. With probability 0.2 the action's effect becomes true and the attacker can additionally perform actions 2:Exploit Vulenrability and 3:Password Brute Force. The attacker obtains reward 1000 only in the case that fact 1:execCode(Database,Root) becomes true during an attack.*

We use freely available MulVAL tool to generate the networks logic-based attack graphs. These attack graphs consist of the attacker's *exploit* actions that target a specific vulnerability in a host and *pivot* (or *hop*) actions that allow the attacker to execute exploits on the hosts visible from the already compromised host. Successfully executing an exploit

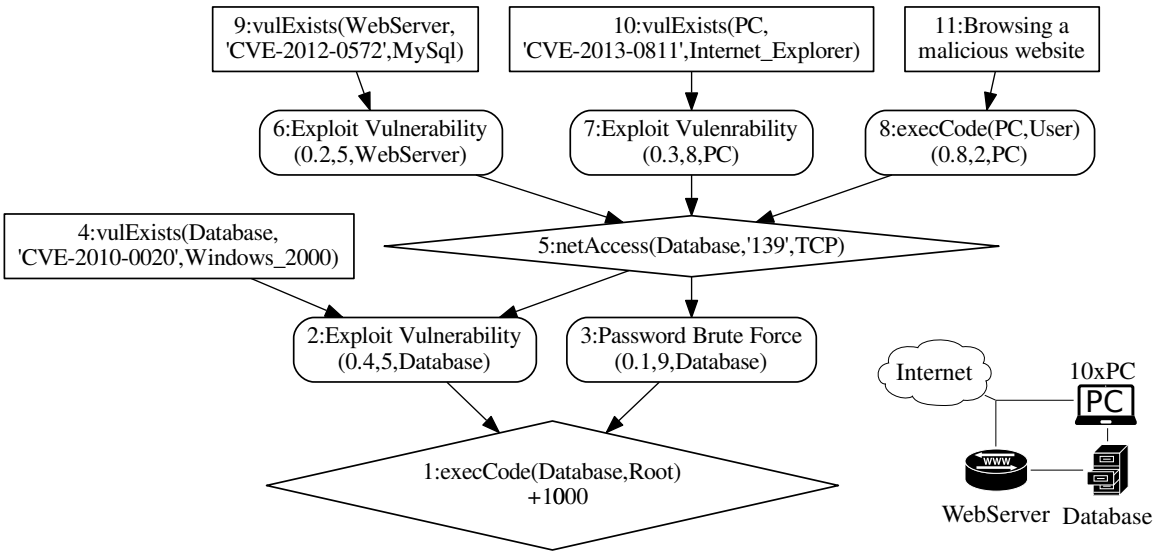


Figure 4.1: A network (right) and an attack graph (left), that represents the ways to gain the access to the database

results in the attacker gaining *privileged* or *non-privileged access* to that host. Previous works (e.g., [Sawilla and Ou, 2008]) show that the information about the costs and success probabilities for different actions can be estimated using the Common Vulnerability Scoring System ([Mell et al., 2006]) values available for example in the National Vulnerability Database ([Bacic et al., 2006]), historical data, red team exercises, or be directly specified by the network administrator.

Correlated Actions The actions in the attack graph are currently assumed to be uncorrelated, which means that performing one action does not change the cost or the probability of any other action. This assumption tremendously simplifies the optimal strategy planning. This is similar assumption to naive Bayes in the classification problems. However, in reality the actions may be correlated: when the attacker performs e.g. scanning of the network it could change his estimation about the cost or probability of a certain action that he knew he could have performed earlier anyway, but with different cost or probability. Nevertheless, obtaining the actions' relationships is a difficult task, as there are $N(N-1)$ direct relationship couples, where N is a total number of actions. Given that there are couple of hundred actions, this task becomes infeasible by to manage by hand. However, there might be some relationship between the actions based on their preconditions and effects, from which the actions' correlation could be computed.

Action Probabilities The basis for the optimal policy planning is the attack graph structure, action costs and probabilities. The MulVAL tool mainly focuses on creating good attack graph structure. It also provides an option to calculate the action probabilities, however, in very simplistic way. In particular, MulVAL offers two options: (i) "with metric artifact", which statically assigned to every action rule one of the following probabilities: 1.0 (certain), 0.8 (likely), 0.5 (possible) or 0.2 (unlikely), or (ii) "with metric", which sets

probability based on a possible values of Access Vector (high, medium or low) from Common Vulnerability Scoring System[1] as $\{\text{high} \to 0.9, \text{medium} \to 0.6, \text{low} \to 0.2\}$.

However, in our experience these options are too limiting and all the probabilities often result in bimodal values, e.g., for (ii) either high or low. Therefore, in our experiments we sample the probabilities from the normal distribution to simulate the lager and more realistic spectrum of probability values.

Action Costs represent the attacker's effort for attempting to perform an exploit. There are few reasonable cost interpretations, e.g., duration of the exploit (this metric is useful for penetration testers who maximize the number of successful attacks during the limited time of period), communication complexity between the attacker and networks (the lower communication complexity, the lower the probability of being detected with e.g., network anomaly detectors) or monetary (amount of money that the attacker pays for obtaining the exploit code). Extracting the relevant information from these sources to calculate the attacker's action costs can be done using machine learning techniques. We leave it as a future work and focus primarily on game-theoretic aspects of the problem.

Pseudorules The MulVAL tool contains also actions, that do not represent any action to perform in reality, but merely the attackers knowledge. Example of such action is the action *canAccessHost* or *canAccessFile*. These actions have the success probability set to 1.0 (certain) by MulVAL. However, this knowledge costs nothing the attacker and has no probability of interacting with honeypots thus we have set both, their costs and the probability of interacting with honeypot, to zero.

4.3.1 Attack Graph Updates

Notice, that only unreached (or false) facts (diamonds in Figure 4.1) restrict the attacker choice of possible exploits that can be performed. True facts, that have been already reached (rectangles in Figure 4.1), can be omitted in the attack graphs without changing its semantic. Let g be an attack graph. After an action is performed (successfully or not) the attack graph can be updated using the following rules:

Rule 1 (remove true facts): If for any fact $f \in F$ in $g : \mathit{pre}(f) = \emptyset$, then remove f from g. This rule removes all the facts from g that are already true (they have no preconditions) and their removal does not change attack graph semantic.

Rule 2 (remove facts with no effects) If for any $f \in F$ in $g : \mathit{eff}(f) = \emptyset$ and $r_f = 0$, then remove f from g. This rule removes all the facts from g that the rational attacker would not want to attack. These are the facts that have no reward for the attacker nor they belong into the precondition of any action.

Rule 3 (remove unnecessary actions): If for any $a \in A$ in $g : \mathit{eff}(a) = \emptyset$, then remove a from g. This rule removes all the actions from g that have empty set of effects. Rational attacker has no incentive to perform them, since all rewards are assigned to facts.

[1] www.first.org/cvss

Rules 1–3 remove facts and actions that the rational attacker would never perform. We define function AG-CLEAN(g), which sequentially modifies g using the rule as long as they are applicable, and returns modified attack graph.

If action a is performed successfully, Algorithm 4 is called, which removes fact a and all the facts, that became true due to action a. If action a fails, Algorithm 5 is called, which (i) removes action a; (ii) removes set of facts F' that can never become true anymore, because there is no alternative sequence of actions, besides a, that could have made them true, and finally; (iii) it removes actions that can never be performed, because at least one of their preconditions are in F'.

Algorithm 4 Action a succeeded.

```
1: function Succ(g, a)
2:     A ← A \ {a}
3:     for all f ∈ eff(a) do
4:         F ← F \ {f}
5:     end for
6:     g' ←AG-CLEAN(g)
7:     return g
8: end function
```

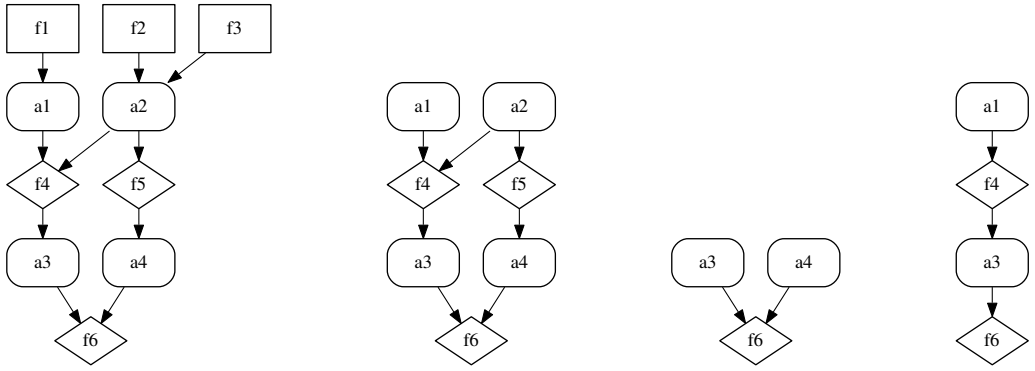
Algorithm 5 Aciton a failed.

```
1: function Fail(g, a)
2:     A ← A \ {a}
3:     for all f ∈ eff(a) do
4:         if ∄a ∈ A : f ∈ eff(a) then
5:             F ← F \ {f}
6:             R = {a ∈ A : f ∈ pre(a)}
7:             for all a' ∈ R do
8:                 Fail(a')
9:             end for
10:        end if
11:    end for
12:    g' ←AG-CLEAN(g)
13:    return g
14: end function
```

Example 4.3.2. *Let us show a simple use examples of the rules over the attack graph in Figure 4.2. Assume the initial attack graph as depicted in Figure 4.2a. Applying Rule 1 results in the attack graph in Figure 4.2b. Next, assume that action $a2$ is performed successfully in Figure 4.2b and Succ($g, a2$) is called. This function removes only facts $f4$ and $f5$, but Rule 3 additionally removes action $a1$. If action $a2$ is performed and fails, function Fail($g, a2$) is called. (i) It removes $a2$; (ii) since there is no other action that could make $f5$ true, $f5$ can be removed; and (iii) consequently it removes action $a4$, which contains at least one infeasible fact ($f5$) and the final attack graphs is depicted in Figure 4.2d.*

4.3.2 Attack Policy

In this section, we formally introduce the attacker's *contingent attack policy* over an attack graph that fully characterize the attacker's behavior. It prescribes the attacker's next action in case the previous action failed or succeeded. This allows identifying not only the actions likely to be executed by a rational attacker, but also the *order* of their execution. This is necessary to evaluate effectiveness of deploying honeypots or any other intrusion detection sensors. Figure 4.3 depicts the optimal attack policy for the attack graph in Figure 4.1. Here, the attacker first attempts to perform action *8:execCode(PC,User)*. If the action is



(a) (b) (c) (d)

Figure 4.2: (a) Initial attack graph. (b) Attack graph after removing true preconditions. (c) and (d) are attack graphs after action $a2$ succeeds (resp. fails) in attack graph from (b).

successful, he follows the success (solid edge) subpolicy, thus performing action *2:Exploit Vulnerability*, otherwise the fail (dashed edge) subpolicy is followed, starting with action *6:Exploit Vulnerability*, and so on.

Formal Attack Policy Definitions

Definition 13 (Attack policy)**.** *An* attack policy ξ *is an oriented binary tree* (V, E)*, where* V *are vertices and* $E = E^+ \cup E^-$ *are edges. Each vertex* $v \in V$ *is mapped to an action* $action(v) \in A$*, and each edges* $e \in E$ *represents either success* ($e \in E^+$) *or failure* ($e \in E^-$) *of the parent action and* E^+ *and* E^- *are disjoint.*

Definition 14 (Attack path)**.** *A* $path(\xi, v_l) = (v_1, e_1, \ldots, v_{l-1}, e_{l-1})$ *is an ordered sequence of vertices and edges in attack policy* ξ *from the root to some vertex* v_l *(excluding node* v_l*).*

Definition 15 (Executable action)**.** *Action* a *is* executable *if and only if its preconditions are satisfied.*

Let us define useful notation for attack path *path*. Let $V(path)$ be the set of all vertices along the *path*, and $V^+(path) \subseteq V(path)$ (resp. $V^-(path) \subseteq V(path)$) be the set of vertices that preceded a successful (resp. unsuccessful) edge. Let $\mathit{eff}^+(path) = \cup_{v \in V^+(path)} \mathit{eff}(action(v))$ be the set of effects that became true along the *path*.

Definition 16 (Valid path)**.** *A path is* valid *if and only if all actions along the path are executable.*

Definition 17 (Valid attack policy)**.** *Attack policy* $\xi = (V, E)$ *is* valid *if and only if all attack paths in it are valid.*

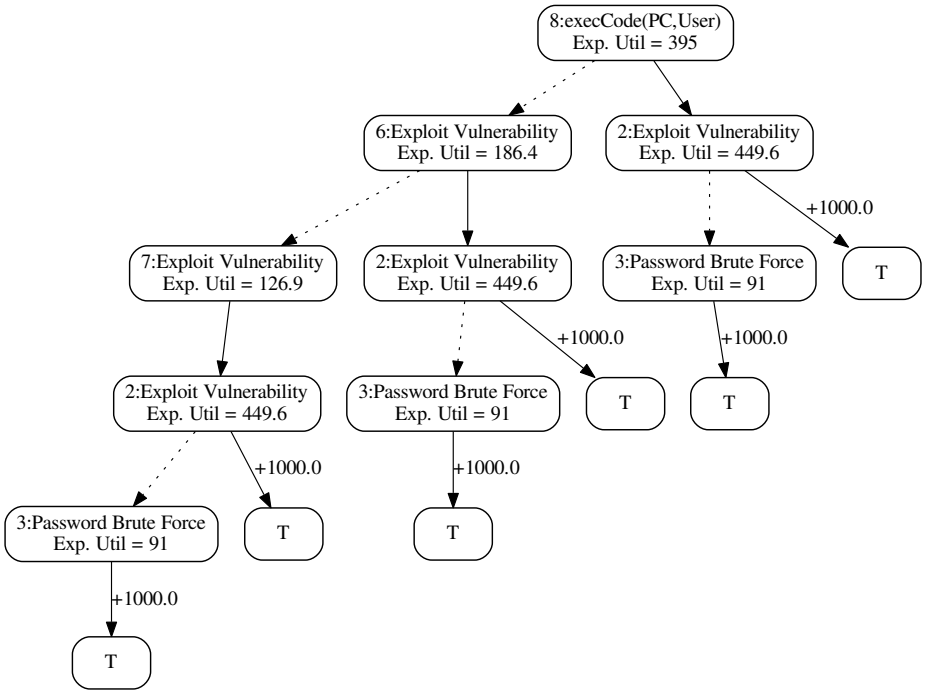


Figure 4.3: Optimal attack policy for the attack graph in Figure 4.1. The policy begins with root action no. 8. If the attempted action succeeds (resp. fails), then solid (resp. dashed) edge is followed. At each stage of the policy the attacker's expected utility is denoted. T denotes the termination action.

Let ξ_v be a subtree of valid attack policy ξ rooted at vertex v with action $a = action(v)$. The *expected utility* of an attack policy ξ on attack graph g is defined recursively as follows:

$$U(\xi_v, g) = p_a(U(\xi_{v^+}, g) + IR(\xi_v, g)) + (1 - p_a)U(\xi_{v^-}, g) - c_a \tag{4.1a}$$

$$= \underbrace{p_a(R(\xi_{v^+}, g) + IR(\xi_v, g)) + (1 - p_a)R(\xi_{v^-}, g)}_{R(\xi_v, g)} + \tag{4.1b}$$

$$\underbrace{p_a C_g(\xi_{v^+}, g) + (1 - p_a)C_a(\xi_{v^-}, g) - c_a}_{C(\xi_v, g)} \tag{4.1c}$$

$$= R(\xi_v, g) + C(\xi_v, g) \tag{4.1d}$$

where ξ_{v^+} (resp. ξ_{v^-}) is the subtree of ξ_v after action a succeeds (resp. fails). $IR(\xi_v, n) = \sum_{f \in \mathit{eff}(a) \setminus \mathit{eff}^+(path(\xi, v))} r_f$ is an immediate reward for successfully performed action a. It can be decomposed into the attacker's *expected reward* $R(\xi_v, g)$ (in Equation 4.1b) and *expected cost* $C(\xi_v, g)$ (in Equation 4.1c), which we will further make use of. Let $\mathbb{A}(g)$ denote the set of all valid attack policies for attack graph g. The set $\mathbb{A}(g)$ is finite because the attacker never performs the same attack action more than once and on each host

the attacker can perform a finite set of exploits. $\xi^* \in \mathbb{A}(g)$ is *optimal attack policy* if $\forall \xi \in \mathbb{A}(g) : U(\xi^*, g) \geq U(\xi, g)$.

4.4 Honeypot Selection Game Model

Let us now formally define Honeypot Selection Problem (HSP) as follows:

Definition 18 (Honeypot Selection Problem (HSP))**.** *Let be given a set of networks $\mathcal{A}_c$, their probability distribution $\mathcal{C}$, host type values r_t of type t, and the defender's cost $c_d(t)$ for deploying honeypot of type t. Find strong Stackelberg equilibrium solution profile (δ_d, π_a), where the defender's strategy δ_d consists of honeypot allocations (number of honeypots $h_{(n,\tau)} \in \mathbb{N}$ of type τ to allocate into network $s \in \mathcal{A}_c$) and the attacker plays best-response strategy π_a, which prescribes an attack policy for every reachable network extended with the honeypot strategy by δ_d.*

In this section, we describe how to represent this problem using game model described in Chapter 3.1 and compute SSE solution profile. Due to algorithmic complexity of the problem, we further introduce heuristic approaches that we use to solve the game.

4.4.1 Chance Actions

Let $\mathcal{A}_c$ denote the set of networks the Chance chooses form, where each network $s \in \mathcal{A}_c$ consists of $s^t \in \mathbb{N}_0$ hosts of type $t \in T$. Chance selects network s with probability $\mathcal{C}(s)$. In example in Figure 4.1, host types are $T = (A, B, router, target)$ and Chance chooses from three networks in $\mathcal{A}_c$ ={(2,0,1,1), (1,1,1,1), (0,2,1,1)} each with probability $\mathcal{C}(s) = \frac{1}{3}$.

4.4.2 Defender's Actions

The defender chooses at most k honeypot and their types to deploy into each network $s \in \mathcal{A}_c$. Formally, the set of all the defender's actions is $\mathcal{A}_d = \{c \in \mathbb{N}_0^T \,|\, \sum_{t \in T} c^t \leq k\}$, where k is the number of available honeypots and c^t is the number of honeypots of type t. Performing action $c \in \mathcal{A}_d$ on network $s \in \mathcal{A}_c$ results in a network with s^t real hosts of type t and c^t honeypots of type t. The attacker's action on host type t interacts with a honeypot with probability $\frac{c^t}{s^t + c^t}$. In the example in Figure 3.1 the defender deploys exactly $k = 1$ honeypot and the set of the defender's actions is $\mathcal{A}_d = \{(2,0,0,0), (1,1,0,0), (0,2,0,0)\}$. Applying all actions in each network results in 9 different networks for the attacker.

Attacker's Actions

An *information set* is a set of indistinguishable networks by the attacker. In Figure 3.1 we depict information sets I_1 through I_5 for the attacker, where he makes his first decision about an attack action. The attacker chooses a full attack policy $a \in \mathcal{A}_a(I)$ in each I, where $\mathcal{A}_a(I)$ is a set of all valid attack policies for an attack graph of a network in I. Note, that the attacker can update his belief about the honeypot placement along the execution of an attack policy (along an attack paths), which must be taken into account

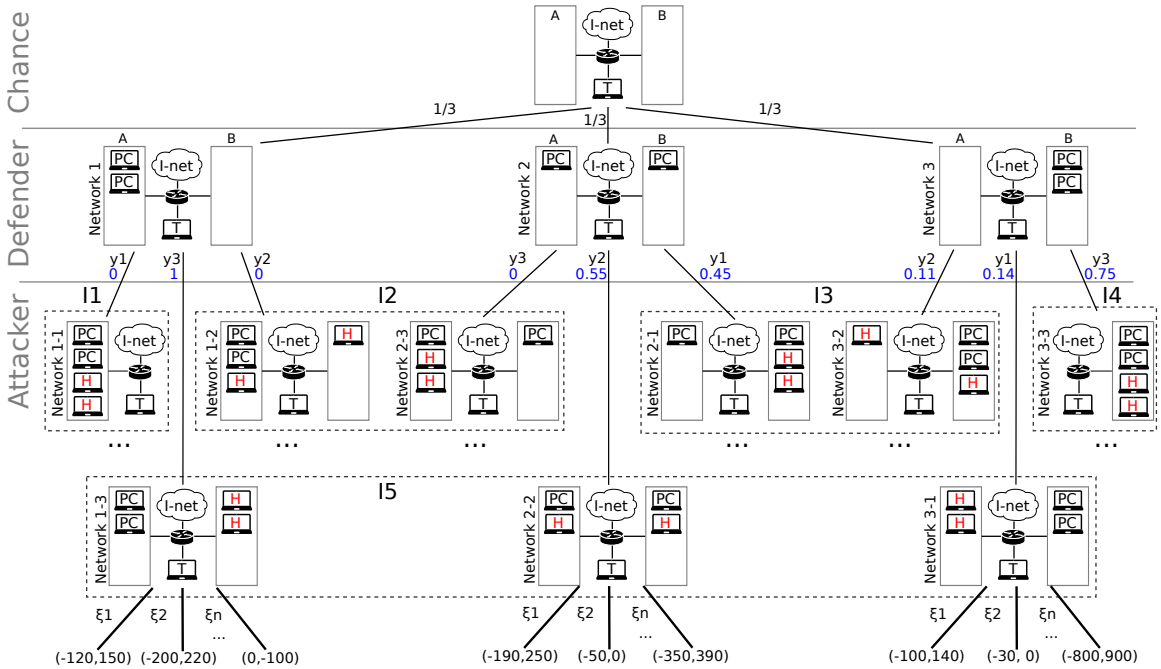


Figure 4.4: The game tree for a simple network with host types A, B, router and target (T). Chance plays uniformly into three possible networks; each contains two hosts of A or B. The defender chooses possible combinations of allocating two honeypots in each possible network; the defender's optimal strategy is depicted with probability values of choosing each action (y_1, y_2 or y_3). The attacker selects attack policy ξ_1 through ξ_n according to the networks' attack graphs

during the optimal attack policy computation. Formally, two states $z = (s_z c_z path(\xi, v_l))$ and $w = (s_w c_w path(\xi', v_k))$ belong to the same information set $I \in \mathcal{I}_a$ if and only if $\forall t \in T : s_z^t + c_z^t = s_w^t + c_w^t$ and $path(\xi, v_l) = path(\xi', v_k)$, where ξ and ξ' are attack policies and v_l and v_k are some nodes in the attack policies, respectively. In words, if two networks after the defender's move result in the same number of hosts of each host type and the attack performed exactly the same attack path on both of them, than the attacker still cannot tell the two networks apart. Networks in states $z, w \in I \in \mathcal{I}_a$ have the same set of vulnerabilities, therefore, the same attack graph structure for the attacker. The only difference is in probabilities of interacting with honeypots.

4.4.3 Players' Utilities

We use r_t to denote value of a host of type t to both players (e.g., value for possessing classified documents in that host). We assume that the defender's cost for deploying honeypot of type t is $c_d(t) = \gamma_c r_t$, which is proportional to r_t by factor of γ_c, but our model is not limited to this. For terminal state $z = s_z c_z a_z \in \mathcal{Z}$, the attacker's expected

payoff in z is $u_a(s_z c_z a_z) = R(s_z c_z a_z) - C_a(s_z c_z a_z)$ and the defender's expected utility is $u_d(s_z c_z a_z) = -R(s_z c_z a_z) - C_d(s_z c_z)$, where:

1. $R(s_z c_z a_z)$ – is the attacker's expected reward for executing policy a_z on network s_z with honeypots deployed according to c_z. This component can be viewed as a zero-sum component (the more the attacker gains the more the defender loses).
2. $C_d(s_z c_z a_z) = \sum_{t \in T} c_d(t) c_z^t$ – the defender's cost for deploying honeypots according to c, where c_z^t is the number of honeypots of type t.
3. $C_a(s_z c_z a_z)$ – the attacker's cost for executing policy a_z on network s_z with honeypots deployed according to c_z.

The values for $R(s_z c_z a_z)$ and $C_a(s_z c_z a_z)$ we compute equivalently to Equations (4.1b) and (4.1c). The only difference is in the case that action a_z interacts with host type t for the first time along its attack path as described in formulas. Then with probability $\frac{c_z^t}{s_z^t + c_z^t}$ the attack ends and the attacker receives zero, and with probability $\frac{s_z^t}{s_z^t + c_z^t}$ the attack continues. In Figure 3.1 the first value is the defender's utility and the second value is the attacker's utility.

In strategy profile (δ_d, δ_a) the defender's expected utility is $u_d(\delta_d, \delta_a) = -R(\delta_d, \delta_a) - C_d(\delta_d, \delta_a)$, where the first component is the defender's expected loss and the second component is the expected honeypot deployment cost, computed from expected payoffs in the terminal states of the game reached by the strategy profile. The attacker's expected utility is $u_a(\delta_d, \delta_a) = R(\delta_d, \delta_a) - C_a(\delta_d, \delta_a)$, where $R(\delta_d, \delta_a)$ is the expected reward and $C_a(\delta_d, \delta_a)$ is the expected policy execution cost, which depends on both, the defender's and the attacker's strategies. For example, if the attacker interacts with a honeypot at the beginning of the attack, the attacker is detected and does not pay for executing the remaining of the attack policy.

Touching Honeypot

In our work, we assumed that the defender immediately blocks the attacker's activity when the attacker touches the honeypot. However, there are more possibilities how the defender could react, i.e., the more sophisticated defender might observe the attacker's activity to learn about his intentions or to even trace him back to find out how did the attacker got into the network (if the interacted honeypot is not immediately accessible from the Internet, but lies in the inner structure of the network) in the first place. The simple blocking of the attacker leaves him the knowledge how he reached to the honeypot and which types of machines contain a honeypot. So we can assume that the attacker knows how to reach the state right before he performed the action that interacts with the honeypot. Nevertheless, in this point could perform the action again, hoping that this time it would be on a real machine instead of the honeypot. We could model this type of attacker by allowing him to repeat the action and paying some penalty each time he is detected by the honeypot, which is a reasonable assumption. There are various ways of setting the penalty, i.e., proportionally to cost of actions he performed just before interacted with the honeypot, as he has to perform them again (with the different IP). However, there are other options for the attacker, i.e., having bounded number of repetitions of each action.

4.5 Heuristic Approaches

Computing Stackelberg equilibria of imperfect information games with stochastic events is in general case NP-hard, as shown in [Letchford and Conitzer, 2010]. The state-of-the-art algorithm for solving this class of games, presented in [Bošanský and Čermák, 2015], uses mixed-integer linear programming (MILP) and the sequence-form representation of strategies. Our case of attack graph games is also hard because there is exponential number of attack policies for the attacker to consider with the natural parameters that characterize the size of a network. Additionally, the set of actions for the defender grows combinatorially with increasing number of honeypots k, which further limits the scalability of algorithms. Therefore, we focus here on a collection of *heuristic* algorithms that find strategies close to SSE in polynomial time in the size of the game tree. These algorithms compute the optimal solution profiles of different solution concepts or pose more restrictive assumptions on the game. We present the general idea of several heuristic methods first, and then discuss the specific details of the algorithms.

4.5.1 Perfect Information Heuristic (PI)

A straightforward game relaxation is to remove the attacker's uncertainty about the actions of the Chance player and the defender, which results in a perfect information game. Although the authors in [Letchford and Conitzer, 2010] showed that in general, the PI game with chance nodes is still NP-hard to solve, the structure of our game allows us to find a solution in polynomial time. the Chance player acts only once and only at the beginning of a game. After the Chance's move the game is a PI game without chance nodes, which can be solved in polynomial time in the size of the game tree, as shown in [Letchford and Conitzer, 2010].

In our game it corresponds to the defender securing each network separately, without exploiting the attacker's imperfect information. In [Durkota et al., 2015b] it was shown that in PI games the defender protects the network by (i) either duplicating the targeted host, if few honeypots are available or (ii) if more honeypots are available, then duplicating the host types at the "choking points" of a network (hosts, through which the attacker must always pass to compromise the network, e.g., all entry nodes into the network). This approach is likely to find these suboptimal strategies.

4.5.2 Zero-Sum Heuristics (ZS)

In [Korzhyk et al., 2011] the authors showed that under certain conditions approximating the general-sum (GS) game as a zero-sum (ZS) game can provide an optimal strategy for the GS game. In this section, we use a similar idea to construct ZS game heuristic, for which we compute a NE that coincides with SSE in ZS games. However, in our case this approach does not provide any optimality guarantees of the results as in [Korzhyk et al., 2011], but it may result in strategies close to the optimal SSE. A Nash equilibrium can be computed in polynomial time in the size of the game tree using the LP from [Koller et al., 1996].

Recall that in our game the defender's utility is $u_d(sca) = -R(sca) - C_d(sca)$ and the attacker's utility is $u_a(sca) = R(sca) - C_a(sca)$, where $sca \in \mathcal{Z}$. Since $R(sca)$ is a zero-sum component, the smaller $|C_d(sca) + C_a(sca)|$ is, the closer our game is to a zero-sum game. Therefore, for small costs, we expect this heuristic to compute strategies close to SSE. We propose several variants of ZS game:

- (ZS1) Both players consider only the zero-sum component ($u_d(sca) = -R(sca)$). Both player costs are neglected, which means that the attacker performs attacks and the defender deploys honeypots at no cost. We expect the defender to deploy all the available honeypots prioritizing the valuable host types first.
- (ZS2) Only the attacker's payoffs are considered ($u_d(sca) = R_d(sca) + C_a(sca)$). Since it omits the defender's honeypot deployment costs, he is motivated him to deploy them in such a way, that the attacker's policies become costly. However, it is hard to conclude how the defender will allocate honeypots to achieve this.
- (ZS3) Only the defender's payoffs are considered ($u_d(sca) = R_d(sca) - C_d(sca)$). Here, the defender's honeypot deployment costs are considered, therefore, he will balance their deployment cost with their return in terms of network protection. Since the attacker *cannot* influence the defender's honeypot deployment cost ($C_d(sca)$ depends only on sc and not on a), the attacker is not additionally motivated to increase the honeypot deployment cost to the defender. However, the attacker's attack costs are neglected, therefore, he may perform attacks which may incur loss to him if the costs were regarded.
- (ZS4) The defender keeps his original utility with adversarial motivation to harm the attacker ($u_d(sca) = R_d(sca) - C_d(sca) + C_a(sca)$). Here, the defender is willing to put his network at higher risk if it results in more expensive attack policies for the attacker. However, in reality the defender does not care about the attacker's costs.

The larger the individual costs $C_a(\cdot)$ and $C_d(\cdot)$, the further the original game is from the zero sum game, therefore, the worse strategies the zero sum heuristic should find, as was shown in [Durkota et al., 2015a]. These approaches could perform well in games with relatively small player costs compared to their rewards.

4.5.3 Correlated Stackelberg Equilibrium Heuristic (CSE)

The main motivation for this heuristic is the concept of correlated equilibria and an extension of the Stackelberg equilibrium in which the leader commits to a correlated strategy. In [Conitzer and Korzhyk, 2011] this concept has been used in normal-form games, in [Čermák et al., 2016b] for extensive-form games and in [Letchford and Conitzer, 2010] for stochastic games. In this case, the leader is not only committing to a mixed strategy but also signals to the follower an action that the follower should take such that the follower has no incentive to deviate. By allowing a richer set of strategies, the leader can gain at least the same utility as in the strong Stackelberg equilibrium solution concept. However, the defender's commitment to signal the attacker the action to play impedes this solution concept from being applied in practice, since the defender may not be able to communicate with the attackers.

Computing a correlated equilibrium of normal-form games can be done with polynomially large linear program to the size of the game. But, computing correlated Stackelberg equilibrium in extensive-form games with imperfect information and chance nodes is an NP-hard problem (follows from Theorem 1.3 in [von Stengel and Forges, 2008]) in general. However, we can exploit a specific structure of our game to compose a polynomially large linear program in the size of the game tree. Specifically, because the Chance acts only once, we can represent the game-tree as a set of normal-form games, one for each of the attacker's information sets. Additionally, a global constraint required that connects all normal-form games due to the Chance player probabilities. We describe this algorithm in Section 4.6.3. This solution concept can be used twofold: (i) the defender's expected utility in CSE is an upper bound for the defender's expected utility in SSE, and (ii) the computed defender's strategy may provide a strategy close to SSE, so it can serve as a polynomial heuristic to compute reasonable strategy. In [Durkota et al., 2015a] we used CSE as an upper bound on SSE to measure the maximal error of other heuristics to the optimal SSE. In this thesis, we use CSE to compute the heuristic strategy only.

All three heuristics we use to compute strategies in polynomial time to the size of the game tree. However, developing a reasonable theoretical bounds on the defender's utility loss for playing heuristic strategy instead of the SSE strategy in an imperfect-information game is difficult due to complexity of the problems. All our attempts resulted in very loose bounds, which are practically useless. Therefore, in Section 4.7 we experimentally evaluate the quality of the heuristic strategies and compare them to the SSE strategies.

4.6 Algorithms

In this section, we describe algorithms for computing the exact and heuristic defense strategies for Honeypot Selection Problem. One of the major computational difficulties is in computing the attacker's attack policies for the game. Different heuristic algorithms solve this problem in a different way.

The structure of this section is as follows. First, we present novel algorithms that compute the attacker's attack plans. These algorithms are key components for constructing the games and computing the attacker's best responses. Then, we introduce novel algorithms to compute the exact and heuristic strategies.

4.6.1 Computing Optimal Attack Policy for Single Network using MDP

This algorithm is used primarily in perfect information approximation, since it assumes that the attacker knows exactly the original network and the defender's honeypot deployment action. Despite the fact that attacker *knows* the number of deployed honeypots of each host type in a network, by attacking a host type he cannot influence which particular host of that type to attack. This algorithm computes the attacker's optimal attack policy using several pruning techniques. These techniques are further reused in other algorithms that compute the attacker's best-response strategies.

We represent the problem of computing the optimal attack policy for an attack graph $g = \langle T, F, A, p_a, c_a, \tau_a, r_f \rangle$ as a finite-horizon Markov Decision Process [Bellman, 1956]. The MDP is defined as a tuple $\langle A_{MDP}, S_{MDP}, T, R \rangle$, where:

- $A_{MDP} = A \cup \{a_T\}$ is the set of in the attack graph with additional termination action a_T.
- S_{MDP} is the set of states $s = (g_s, \tau_s)$, where g_s is a possibly modified attack graph g, and $\tau_s \subseteq T$ is the set of host types that the attacker has interacted with so far. The initial MDP state is $s_0 = (g, \emptyset)$.
- $\mathcal{T}[s'|a, s]$ is the probability that performing action a in state s leads to state s'. Performing action a may lead to three possible states. (i) a interacts with a honeypot with probability $h = 1 - \prod_{t:\tau_a \setminus \tau_s} (\frac{s^t}{c^t + s^t})$ and his attack immediately ends (leads to terminal states s_T); (ii) action a does not interact with a honeypot and is successful with probability $p_a(1-h)$, it reaches state $s' = (\text{Succ}(g_s, a), \tau_s \cup \{\tau_a\})$. or (iii) action a does not interact with a honeypot neither is successfully performed with probability $(1-p_a)(1-h)$ and reaches state $s' = (\text{Fail}(g_s, a), \tau_s \cup \{\tau_a\})$. Functions $\text{Succ}(\cdot)$ and $\text{Fail}(\cdot)$ defined in Algorithm 4 and Algorithm 5.
- $R(s, a, s')$ is the attacker's reward for performing action a in state s leading to s'. It is based on the set of facts that became true, the action cost c_a and whether a interacts with a honeypot.

This is a finite-horizon MDP, since the attacker can perform each exploit actions at most once and there are finite set of actions in an attack graph. Thus, we apply Forward Dynamic Programming algorithm from Section 2.1.1 to compute the optimal attack policy. The algorithm is based on the depth-first search with several pruning techniques to speed up the search. In every state $s \in S_{MDP}$ the algorithm computes the expected utility for every executable action from $a \in \alpha_s$, assuming that then the optimal policy is followed. The algorithm constructs the optimal policy by prescribing to state s the action with the highest expected utility. In the case there are more actions with the same expected utility to the attacker, the attacker chooses the one that maximizes the defender's utility by maximizing the defender's reward $R(\xi, z)$ to ensure that it finds strong Stackelberg equilibrium. To speed up the algorithm, we use various techniques which avoid computing the expected utility of the unpromising actions. First, we describe the techniques in high level, and in the remainder of this section in more details.

Dynamic programming significantly enhances the performance of the algorithm. Since the same states in the MDP can often be reached via more than one sequence of actions, we cache the expected rewards and corresponding policies of the visited states and reuse it when possible. E.g., in Figure 4.3 can be seen the repetition of the subpolicies, which we compute only once because of the dynamic programming. The second technique uses **sibling-class pruning** in every state $s \in S_{MDP}$ to prune out the actions that are guaranteed to be suboptimal. Intuitively it states, that if in state s the attacker can perform two actions and both have the same effect, he first executes the action with the highest success probability to cost ratio (the succeed-fast-and-cheap strategy). Similarly, if the attacker must perform two actions to reach fact f, he first executes the action with the highest fail probability to cost ratio (the fail-fast-and-cheap strategy). Third, we apply **branch and bound** techniques to further eliminate not promising actions in state s. For

that, we compute *lower* and *upper bounds* of the expected utility of each action in s. If the upper bound of action a is lower than the lower bound of action b, we prune out a.

Sibling-Class Pruning

The complexity of the optimal attack planning originates in the need to explore all possible orderings of actions. The sibling-class theorem states that in some situations, the optimal order for executing actions can be determined directly without any search. This theorem was proven in [Greiner et al., 2006] in the context of "probabilistic AND-OR tree resolution" (PAOTR). The AND part of the theorem is proven in the context of simplified attack graphs in [Buldas and Stepanenko, 2012]. In both of these works, the actions can be present only in the leafs of the AND-OR graph, without any preconditions. The inner nodes represent only the conjunctions and disjunctions and do not have any costs or probabilities of success. Moreover, the theorem in [Greiner et al., 2006] is proven only for AND-OR trees in which no node can have more than one parent. There is no notion of reward, and the AND-OR tree is evaluated until the root of the tree is resolved.

In the attack graphs generated by existing network security tools (e.g., MulVAL [Ou et al., 2006]), attack actions are present in the inner nodes of the attack graphs, which can also be cyclic graphs. Furthermore, we assume that the attacker can obtain intermediate rewards for activating subsets of fact nodes and not only for reaching the root of the attack graph. Here we generalize the sibling-class theorem to handle these additional cases. We provide only a proof sketch here based on the original proof. We find that the original proof holds with minor modifications. To describe the theorem, we first need to define several additional concepts.

Definition 19. OR-sibling class *is a set of executable actions each of which have the same set of effect facts $f \subseteq F$ in the attack graph and do not have any other effect besides f.*
AND-sibling class *is a set of executable actions $X \subseteq A$ in the attack graph, such that there is a "grandchild" action $a_g \in A$ that can be executed if an only if all of the actions in X are successful ($pre(a_g) \subseteq \bigcup_{a \in X} eff(a) \& \forall b \in X\ pre(a_g) \setminus \bigcup_{a \in X \setminus \{b\}} eff(a) \neq \emptyset$) and none of the actions has an effect that would help enable other actions, besides a_g:*

$$\forall a_1, a_2 \in X; a_g, a'_g \in A eff(a_1) \subseteq pre(j_1) \& eff(a_2) \subseteq pre(a'_g) \Rightarrow a_g = a'_g.$$

Definition 20. *We define the* R-ratio *of actions in sibling classes as*

$$R(a) = \frac{p_a}{c_a} \text{ if action } a \text{ is in OR-sibling class; and} \tag{4.2}$$

$$R(a) = \frac{1 - p_a}{c_a} \text{ if action } a \text{ is in AND-sibling class.} \tag{4.3}$$

The sibling theorem states that actions in the optimal policy for an attack graph that belong to the same sibling class are always executed in decreasing order of their R-ratios.

Theorem 4.6.1 (Sibling class theorem)**.** *Let AG be an attack graph and let ξ^* be the optimal policy for the attack graph. Then for any actions x, y in the same sibling class, such that $R(y) > R(x)$, x is never performed before y in ξ^*.*

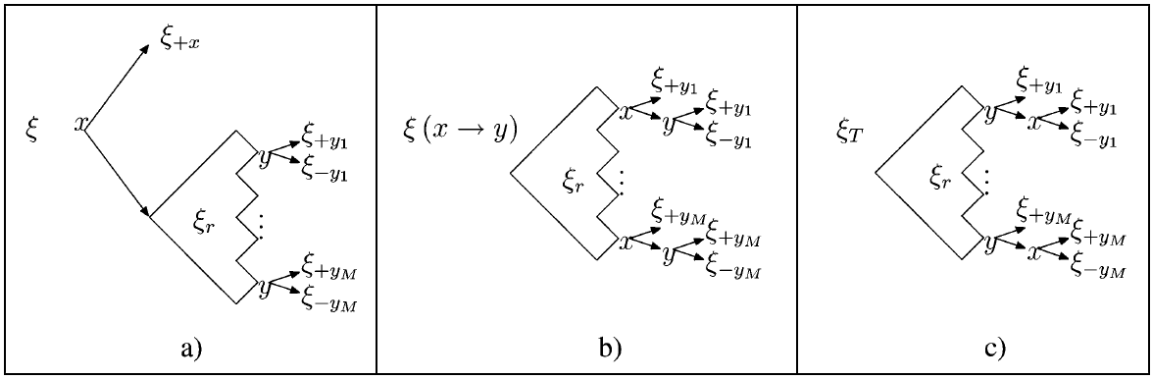


Figure 4.5: Illustration of the proof of the sibling theorem taken from [Greiner et al., 2006]

Proof. The proofs for the AND and OR sibling classes are symmetric; hence, we focus only on the OR sibling class. The proof of this theorem in [Greiner et al., 2006] relies on a lemma about attack policies. If we denote a tree rooted in action x with left subtree ξ_+ and right subtree ξ_- by $(x;\xi_+;\xi_-)$ then the lemma can be rephrased in the following way:

Lemma 4.6.1. *Let $x, y \in A$ be actions in the same OR-sibling class of an attack graph such that $R(y) > R(x)$. Let ξ_{xy} be an optimal policy for the attack graph in the form $(x;\xi_+;(y;\xi_+;\xi_-))$, then $\xi_{yx} = (y;\xi_+;(x;\xi_+;\xi_-))$ is a valid policy with a higher expected reward than ξ_{xy}.*

The proof of the Sibling class theorem in [Greiner et al., 2006] works by induction on the number of attack actions in the optimal policy. It is demonstrated on Figure 4.5 taken form the article. For contradiction, assume that there are actions x and y in the wrong order in the optimal attack policy. Without loss of generality, assume a case with an action x in the root of the policy and its sibling y at several places in the negative branch of x (Figure 4.5(a)). The positive branch does not include y, because it is an OR-sibling of x, which means that the fact achieved by y is already achieved by x, if it is successful. The proof shows that the expected utility of the policy is not decreased if it starts as the negative branch of x and action x is executed just before action y would be executed in the original policy (Figure 4.5(b)). Then the proof uses the Lemma 4.6.1 to show that if each instance of the actions x and y are switched (Figure 4.5(c)), the utility of the policy is increased if the R-ratio of y is higher than R-ration of x.

We argue that a very similar proof is applicable for the more general case of attack graphs defined above. The differences between the model in [Greiner et al., 2006] and our model are:

i) The attack graph is in general a cyclic graph and not a tree.

ii) The attack graph has actions with a probability of success and costs in the inner nodes, and not only in the leaves. This creates preconditions for the actions.

iii) Different attacks give different rewards and it is not necessary to resolve the root node of the attack graph.

iv) The attack terminates when the expected cost of continuing the attack is higher than the expected reward, not when the root of the AND-OR tree is resolved.

The proof is based on properties of the optimal policy with very little reference to the structure of the AND-OR graph. The only use of the AND-OR graph is to assure that after the transformations, the policy is still a legal policy. From the definition of the sibling classes, any action that belongs to a sibling class is a leaf of the attack graph. It does not have any preconditions and it can be executed at any place in the policy. The inner nodes of the attack graph are not moved in the tree and the transformations preserve any preconditions the following actions might have. This means that properties (i) and (ii) do not have any effect on validity of the proof.

The policy in [Greiner et al., 2006] does not have any rewards, only costs for the actions. The proof assumes that each subtree of the policy has an expected cost of execution, but it never requires the costs of the sub-trees to be positive. We can define the expected reward (or negative cost) of a leaf node n as the sum of the rewards acquired in the path $path(\xi, n)$. Afterwards, we can treat this leaf as any other subtree in the proof. This resolves difference (iii).

The last difference between the models is caused by changing the point when the attack stops. The attack stops at leaf n if the expected value of the optimal sub-policy continuing from this node is negative. This is a solely a property of the successors of a node in the policy tree. The transformations used in the proof do not change which actions can be executed below a current leaf in the policy and they do not change what other facts can be activated below the current leaves. Therefore, it cannot be optimal to prolong any branch after the transformations. Furthermore, no attack would be terminated earlier due to the transformations. The attacker stops an attack if the expected reward of some subtree would become negative. The proof initially assumes an optimal strategy and shows that the expected utility of the strategy stays the same when x is moved just before y. If the expected cost of playing the subtree rooted at the new position of x were positive, stopping the attack at that place would increase the utility of the whole strategy, which contradicts the optimality of the original strategy. The same holds after switching the positions of x and y. The original proof states that the utility is increased when the two are switched. If in addition, some of the subtrees would have negative expected reward after the switch. Removing the subtree would increase the expected utility even more, which still means that the original strategy was suboptimal. □

Therefore, when we consider which action to perform at a decision point we can consider only those with the highest R-ratios in each sibling-class and those that have multiple parents or grandparents. After some actions in the attack graph are executed the inner nodes in the attack graph can become leaf nodes and the sibling theorem is then applied to those nodes as well. Unfortunately, the attack graphs with honeypots violate the monotonicity property of actions (assumed in the proof of Sibling Class Theorem). These actions cannot be pruned or preferred to other actions, therefore, we apply the sibling-class theorem only to actions that do not interact with honeypots.

Example 4.6.1. *Let us demonstrated the Sibling-Class theorem on an example in Figure 4.1. Actions number 6, 7 and 8 belong to the same OR sibling class and their respective*

R-ratios are $\frac{0.2}{5} = 0.04, \frac{0.3}{8} = 0.0375$, *and* $\frac{0.8}{2} = 0.4$. *Sibling class theorem states, that if optimal attack strategy begins with an action from this sibling class, then it must start with the one that has maximal R-ratio. In our case, the attacker orders the actions in decreasing R-ration order: first attempts to perform action 8, if fails then 6, and if even that fails then 7.*

Sibling-Class Theorem with Honeypots The global effect of the honeypots can violate the monotonicity property of actions in attack graph that is used to prove the Sibling Theorem presented previously. The results can still be used in some cases, but applying the result is somewhat more complicated because we must take into account the presence of honeypots. Follows an example that demonstrates inapplicability of the Sibling-Class theorem on attack graphs with honeypots.

Example 4.6.2. *Consider an example depicted in Figure 4.6 with two actions a and b. Action a succeeds with probability* $p_a = 1$, *costs* $c_a = 10$ *and interacts with host type* $\tau_a = \{pc1\}$. *Action b succeeds with probability* $p_b = 0.5$, *costs* $c_b = 1$ *and interacts with host type* $\tau_b = \{pc2\}$. *Both types consist of a real host and a honeypot of that type, therefore, both actions have probability of interacting with a honeypot* $h_a = h_b = \frac{1}{2}$. *Since both actions have the same effect, they belong to the same OR-sibling class. Thus* $0.1 = \frac{1}{10} = R(a) < R(b) = \frac{0.5}{1} = 0.5$. *According to the sibling-class theorem it is optimal to first perform action b and then perform action a (strategy* $\xi_{b,a}$*). However, performing the sequence in the opposite order (strategy* $\xi_{a,b}$*) yields higher expected utility. Compare* $\xi_{b,a} = (1 - h_b) \cdot (p_b \cdot 100 + (1 - p_b)((1 - h - a)(p_a \cdot 100) - 10)) - 1 = 34$ *and* $\xi_{a,b} = (1 - h_a) \cdot (p_a \cdot 100) - 10 = 40$*). The reason is that if action b interacts with a honeypot, the attacker will not get a chance to perform action a anymore, that has success probability 1. Thus, it is better for him to start with action a, despite the fact that its R-ratio is lower than of the action b.*

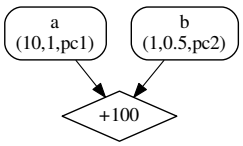


Figure 4.6: Sibling-Class theorem counterexample when actions can interact with honeypots.

Therefore, we use the sibling-class theorem only for actions that do not interact with a honeypot, which is (i) if attacker already interacted with a real host of a type or (ii) if there is no honeypot of that type.

Branch and Bound Technique

The branch and bound needs to compute the lower and upper bounds on the expected utilities of the states. We use lower and upper bounds on the attacker's expected utilities of the states to quickly prune out those, that have upper bounds lower than the lower bound of an alternative reachable states to the attacker. The **lower bound** LB in state s we set to the exact expected utility of some action $a \in \alpha_s$ performed in s. If there is no action for which the exact expected utility has been computed then lower bound is set to zero, which corresponds to the termination action a_T.

The **upper bound** for state s we compute by performing a depth-first search on the attack graph relaxed in two ways. We set action costs to zero, and the probabilities of touching honeypot to zero. One can easily see that this can only increase the attacker's expected utility in the modified attack graph. In the relaxed attack graph for each fact node f with reward $r_f > 0$, we compute the upper bound on the probability that fact f becomes true. The probability is determine by calculating the probability that at least one of the actions in $pre(f)$ will have all its preconditions $pre(a)$ satisfied *and* action a will be performed successfully. This leads to a recursive computation of the probabilities that the action's preconditions in $pre(a)$ becomes true, etc. The algorithm terminates either by reaching the facts that are already true, or reaching the facts that can never be satisfied ($pre(f) \cap A = \emptyset$). To avoid complications caused by cycles, during the traversal we restrict each fact node to be precondition of at most one action in the attack graph. Removing preconditions for the actions can only increase the resulting probability because fewer facts are required to fulfill the preconditions of that action. Let $P_{true}[f|s]$ be the computed upper bound on the probability that fact f will become true in state s. The final upper bound on the expected utility is $\sum_{f \in \{f \in F | r_f > 0\}} P_{true}[f|s] r_t$. We use standard branch and bound technique to prune out provably suboptimal actions.

When we compute the expected utility of a, which was not pruned out, we first compute the expected utility of the success branch of the action. This allows us to compute tighter lower bound for the case the action fails. Namely, we bound it by $LB = \frac{M + c_a - p_a U_a(\xi_{a^+} + IR(\xi_{a^+}))}{1 - p_a}$, where M is the highest expected utility for an action in s for which the exact expected utility has been computed. We used this algorithm for computing the perfect information heuristic, but its pruning techniques we reuse in the other algorithms for computing attacker's attack policies.

Cycles in Attack Graph

The literature usually deals with an acyclic attack graphs, since introduction of the cycles raises certain difficulties to the algorithms or the algorithm may not even terminate. In [Wang et al., 2008b] the authors deal with the three types of cycles, depicted in Figure 4.7. For each type of the cycle they propose how to deal with them: either they can be ignored, broken or must be dealt with in a non-trivial way. The cycle types differ from each other in a way they are entered. We argue, that our MDP approach can solve all mentioned cycles without need of their removal or other procedures.

An example of the first type of cycles is depicted in Figure 4.7a. This cycle can be entered either with action e_1 or e_2. However, both have a precondition which is an effect of the other action. Our MDP approach will never enter this cycle, since action e_1 or e_2 will never have its preconditions true.

An example of the second type of the cycle is depicted in Figure 4.7b, which has an entering point through the fact c_2. In the contrary to the first type, this cycle can be entered. When our MDP approach enters the cycle through c_2, the action e_2 will be removed by AG-CLEAN($\cdot$) function presented in Section 4.3.1, therefore, MDP will not experience the cycling.

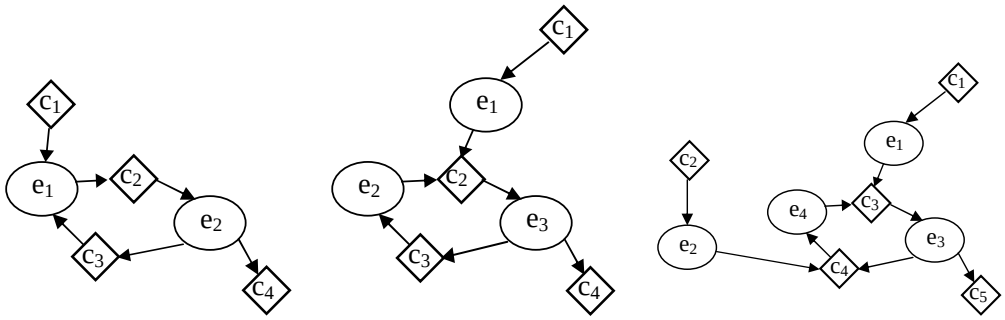


Figure 4.7: Three types of cycles in attack graph from [Wang et al., 2008b].

Finally, the third type, where two (or more) entering points exist, and both are through the fact nodes. According to the authors in [Wang et al., 2008b], this cycle cannot be either removed nor simply broken. For our approach this is not an obstacle either, as our MDP will enter the cycle either from c_4 or c_3. Without loss of generality, let us assume it enters through c_4, which is consequently removed but action e_3 cannot be removed, because it has effect c_5. If e_4 is performed in future, it set true only c_5 (c_4 has been removed), so no cycle appears.

4.6.2 Optimal Attack Policy for Multiple Networks in Information Set

The previous algorithm assumes that the attacker knows precisely the attacked network. In this section, we relax the assumption of perfect observability and introduce two novel algorithms that allows attacker's uncertainty. There are two degrees of uncertainty that the attacker may have: either he *knows* the probability distribution of the networks, or he does not know even that.

Known Probability Distribution Of Networks

Finding the optimal attack policy for a set of networks in an information set I with a known prior probability distribution over them can be solved by extending the previous algorithm. For each network in I we create an MDP as described in Section 4.6.1 with a prior probability of the initial state of in that MDP equal to the probability of that network in an information set. The result is a large POMDP, where the attacker has probability distribution over the initial states. We use similar algorithm as in Section 4.6.1 to solve the POMDP with minor modifications described below.

In Figure 4.8 is an example of the attacker's two MDPs for two networks in information set I_1. The MDPs are created according to the actions in the attack graphs of the networks. E.g., in nodes in I_1 (labeled with a) the attacker chooses an attack action from $A(g)$ (in our example $A(g) = \{a1, a2, a3\}$). After each action, there is a chance that the action interacts with a honeypot (edge h), or does not interact with a honeypot in which case it can succeed (edge s) or fail (edge f). In Figure 4.8 are depicted all edges only for action a_2 at I_1. The

same attack paths lead to the same information sets in the POMDP since the attacker cannot tell them apart. For example, path $(a2, s, a1, f)$ in the left and in the right subtrees in Figure 4.8 leads to the states that belong to the same information set $I_5 \in I$. However, the same action in different MDPs (which are part of the POMDP) may have different transition probabilities, so we use Bayes rule to update the probability distribution among the MDPs based on the action probabilities. The algorithm returns a single attack policy ξ with the highest expected utility *given* the probability distribution over the states in I. During the computation, we use pruning techniques described in the previous section. The above-mentioned algorithm for MDP is modified for POMDP purposes as follows. First, the it keeps the attacker's belief over the MDPs and uses Bayesian rule to update the belief after each action. Second, the algorithm propagates upward expected utility over the MDPs, weighted by the attacker's belief. This algorithm is at least NP-hard since finding the optimal attack policy for one MDP is already NP-hard.

Unknown Probability Distribution Of Networks

In the case the attacker does not know the probability distribution of the networks, there is no specific best-response attack policy that the attacker should play. We compute the smallest set of undominated strategies (SUS) for the attacker that could be best responses

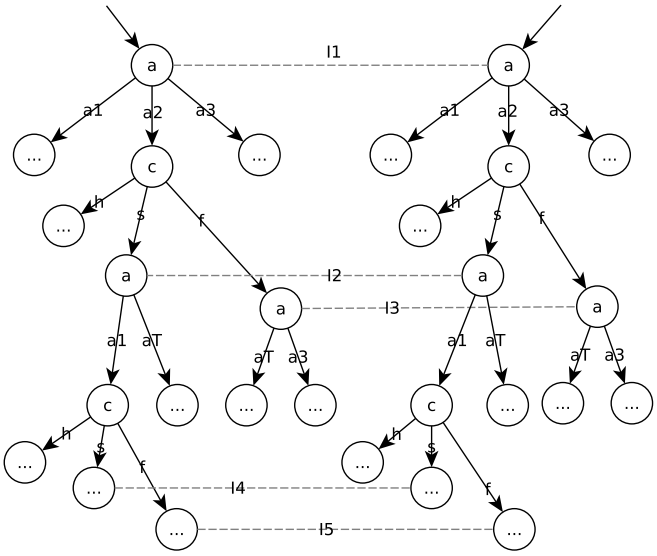


Figure 4.8: The attacker's POMDP consisting of two MDP's, each for a different state in information set I_1. Attacker acts in decision nodes labels with "a" according to the attack graphs of the networks; and the Chance player acts in nodes labeled with "c". After each of the attacker's actions, the Chance determines whether the action: interacted with a honeypot (h), did not interact with a honeypot and succeeded (s) or failed (f).

in I. It significantly reduces the attacker's possible attack actions for further computations of SSE.

We reuse notation $\kappa_a(I)$ to refer to the smallest set of undominated strategies (SUS) for the attacker in I. To compute $\kappa_a(I)$ we use a variant of the *incremental pruning* algorithm [Cassandra et al., 1997], a backward induction algorithm, which in every attacker's decision state propagates the SUS upwards. The Algorithm 6 computes attacker's SUS $\kappa_a(I)$ in information set I.

Algorithm 6 SUS computation for information set I.

```
1: function κ_a(I)
2:     S ← ∅
3:     for all a ∈ A(I) do
4:         I_{a,succ} ← perform(I, a, succ)
5:         I_{a,fail} ← perform(I, a, fail)
6:         SUS_s ← κ_a(I_{a,succ})
7:         SUS_f ← κ_a(I_{a,fail})
8:         for all ξ_1 ∈ SUS_s do
9:             for all ξ_2 ∈ SUS_f do
10:                ξ' ← create new policy a_{(ξ_1,ξ_2)}
11:                S ← S ∪ {ξ'}
12:            end for
13:        end fore
14:    end for
15:    ξ_∅ ← create new policy a_{T(∅,∅)}
16:    S ← S ∪ {ξ_∅}
17:    S' ← ExactSUS(S)
18:    return S'
19: end function
```

Function perform$(I, a, succ)$ (resp. perform$(I, a, fail)$) returns the information set reached by the attacker after action a is successfully (resp. unsuccessfully) performed in I. $a_{(\xi_1,\xi_2)}$ creates a new policy that begins with action a and follows with ξ_1 (resp. ξ_2) if a succeeds (resp. fails). For each action $a \in A(I)$ the algorithm computes SUSs for the case that the action succeeds (SUS_s) and the case it fails (SUS_f). Then, it creates a set of policies S as a combination of the policies that begins with action a and continues with $\xi \in SUS_s$ if a succeeds and with $\xi' \in SUS_f$ if a fails. This can create policies that do not belong to SUS of I, therefore, function ExactSUS$(\cdot)$ is called to eliminate the policies from S that do not belong to SUS, as described in the next section.

Function ExactSUS(S)

Let $U_a(\xi, z)$ be the attacker's utility for choosing policy $\xi \in S$ in state $z \in I$. Here, we apply linear program described in Section 2.3.5 to determine which strategies do not belong to SUS. If there exists a probability distribution over states in I, such that ξ yields maximal

expected utility to the attacker with regards to all policies in S, then ξ is undominated and $\xi \in \kappa_a(I)$. For each policy $\xi \in S$ we use linear feasibility program (LFP) to determine if there exists a probability distribution over states in I, such that ξ yields maximal utility (from [Benisch et al., 2010]):

$$\text{s.t. :}(\forall \xi \in S \setminus \{\xi^*\}) : \sum_{z \in I} U_a(\xi^*, z) p_z \geq \sum_{z \in I} U_a(\xi, z) p_z \tag{4.4a}$$

$$(\forall z \in I) : 0 \leq p_z \leq 1 \tag{4.4b}$$

$$\sum_{z \in I} p_z = 1 \tag{4.4c}$$

The only variables are p_z representing the probability of state $z \in I$. The LFP finds probabilities p_z over I, such that policy ξ is the best response. If the LFP does not have a feasible solution, then policy ξ is never the best response and can be removed from S, otherwise it is kept in S.

Algorithm 7 computes the exact SUS. For each action it runs function SolveLP- 4.4 which tests if the action is undominated. Algorithm 7 already eliminates the vast majority of dominated attack policies, however, in some cases SUS may still contain numerous policies. During our experiments, we encountered the case, where the SUS resulted in many policies that were undominated in a very small probability region or even just in a single point. Removing these policies introduces often insignificant errors to the players. Next, we present an approach to further reduce strategies that introduce at most ϵ error either player.

Algorithm 7 Computes exact SUS given set of policies S.

```
1:  function ExactSUS(S)
2:      for all ξ ∈ S do
3:          κ_a ← ∅
4:          LP ←SolveLP-4.4(ξ, S)
5:          if ∃ feasible solution then
6:              κ_a ← κ_a ∪ {ξ}
7:          end if
8:      end for
9:      return κ_a
10: end function
```

Algorithm 8 Compute error-bounded SUS given set of policies S and error ϵ.

```
1:  function ErrorBoundedSUS(S, ε)
2:      κ̂_a(I) ← S
3:      for all ξ ∈ κ̂_a do
4:          ε_a ← AttackerError(ξ, κ̂_a)
5:          if ε_a ≤ ε then
6:              for all ξ' ∈ κ̂_a \ {ξ} do
7:                  ε_d ← DefenderErr(ξ, ξ', κ̂_a)
8:                  ε_d^max = max{ε_d^max, ε_d}
9:              end for
10:             if ε_d^max ≤ ε then
11:                 κ̂_a ← κ̂_a \ {ξ}
12:             end if
13:         end if
14:     end for
15:     return (κ̂_a, ε_a, ε_d^max)
16: end function
```

Error-Bounded SUS

In order to further reduce the attacker's action space, we allow the attacker and the defender to make a small ϵ error. Intuitively, if removing strategy ξ will not cause a utility difference more than ϵ for either player, then we can remove it from $\kappa_a(I)$. This is similar to weaker-than-weak elimination of dominated strategies studied for Nash equilibrium in [Cheng and Wellman, 2007]. We will denote with $\hat{\kappa_a}(I) \subseteq \kappa_a(I)$ the set of policies that remain after removing all policies from $\kappa_a(I)$ that incur smaller then ϵ error to both players.

Before the technical explanation of the algorithms, let us present the intuition behind the error computation over the example in Figure 4.9. Both plots depict information set $I = \{z_1, z_2\}$, where the x-axis is the probability distribution over I. Y-axis represents the attacker's utility U_a (top plot) and the attacker's reward R (bottom plot). The plots present the attacker's utilities for four different attack policies applied over different beliefs about the states in I. Now, let us analyze how the removal of the strategy ξ^* impacts the players' utilities. If ξ^* is removed, the attacker will play the second best action on the region, where ξ^* was undominated (labeled as Region 1). The attacker can experience at most ϵ_a error, which is the maximal difference between ξ^* and an alternative best response action on the Region 1. To conclude the maximal defender's error, we analyze only the R component of his utility, since the attacker's actions have no impact on the defender's honeypot deployment costs C_d. If ξ^* is removed, we know that the attacker will play either ξ' or ξ''. To determine the error for the case that ξ' is played, we find the maximal difference in R between the ξ^* and ξ' over the intersection of the regions where ξ^* was undominated (Region 1) *and* where ξ' is the attacker's best response after ξ^* is removed (Region 2). Similar computation for ξ'' results in ϵ'' error. After computing errors for all the attacker's alternatives, the defender can experience at most the error equal to the maximum of all errors, in our case $\epsilon_d^{max} = max\{\epsilon'_d, \epsilon''_d\}$. Let us describe the algorithms that determine whether ξ^* can or cannot be removed.

Algorithm 8 iterates over all policies and returns error-bounded SUS $\hat{\kappa_a}(I)$. For each strategy $\xi \in \kappa_a(I)$ the algorithm calculates the attacker's maximal loss ϵ_a due to removal of policy ξ (line 3). If ϵ_a is less then or equal to a given ϵ, then strategy ξ becomes a candidate for removal from $\kappa_a(I)$. All alternative policies $\kappa_a(I) \setminus \{\xi\}$ must be considered (a for-loop in line 5), on region where ξ was undominated. In lines 6 and 7 the algorithm computes the maximum utility difference ϵ_d^{max}. If $\epsilon_d^{max} \leq \epsilon$, then ξ can be removed This algorithm uses linear programs, described below, as a subrutines to compute the players' errors.

Algorithm 8 in line 15 returns set of policies $\hat{\kappa_a}(I)$, and the attacker's and the defender's maximal errors ϵ_a and ϵ_d^{max}, respectively. The errors are propagated upwards in the POMDP to the earlier information set, where they are taken into account to determine the errors of the policies in that information set. For example, assume that some strategies were removed from I_4 and I_5 in Figure 4.8, which introduced errors to both information sets. Now, if a policy recommends to the attacker to performs action a_1 in I_2, then the errors from I_4 and I_5 are reflected in this action. Let $\epsilon_a(I)$ and $\epsilon_d(I)$ denote the attacker's and defender's maximal errors in I. If action a is successfully (resp. unsuccessfully) performed in I which leads to I' (resp. I''), then the propagated error to I is $\epsilon_a(\xi, I) = p_a\epsilon_a(I') + (1 - p_a)\epsilon_a(I'')$, where p_a is the success probability of action a and ξ

is any policy that begins with action a. For the defender we compute the errors analogically with $\epsilon_d(I)$ values. These errors must be also considered when the error for the policy removals are computed,

Function AttackerError($\cdot$) In Algorithm 8, function AttackerError($\xi^*, \kappa_a(I)$) computes the attacker's maximal error by building and solving Linear Program 4.5 as follows:

$$\text{max: } \epsilon_a \tag{4.5a}$$

$$\text{s.t.: } (\forall \xi \in \kappa_a(I) \setminus \{\xi^*\}) :$$

$$\sum_{z \in I} U_a(\xi^*, z) p_z + \epsilon_a(\xi^*, I) - \epsilon_a \geq \sum_{z \in I} U_a(\xi, z) p_z - \epsilon_a(\xi, I) \tag{4.5b}$$

$$(\forall z \in I) : 0 \leq p_z \leq 1 \tag{4.5c}$$

$$\sum_{z \in I} p_z = 1 \tag{4.5d}$$

In this LP, constraint (4.5b) restricts the probability space over I to an upper bound region (due to the errors), where ξ^* could be undominated. It maximizes the difference ϵ_a that the attacker receives for choosing ξ^* and all other policy in $\kappa_a(I) \setminus \{\xi^*\}$ over that region. The objective function ϵ_a contains the attacker's error if ξ^* is removed *including* the errors $\epsilon_a(\xi^*, I)$ and $\epsilon_a(\xi, I)$.

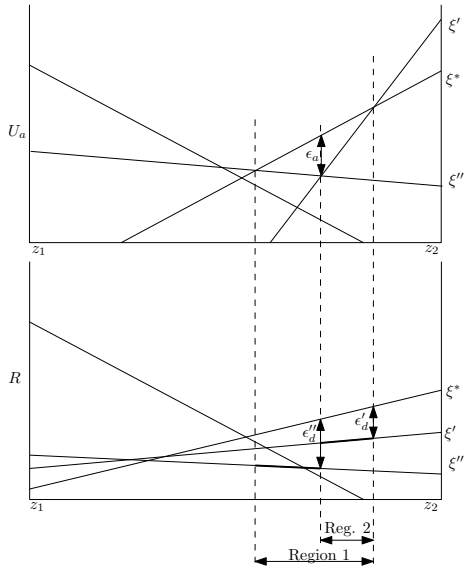


Figure 4.9: Illustration how linear programs 4.5 and 4.6 work. X-axis corresponds to probability distribution between z_1 and z_2. U_a and R_d is the attacker's utility and the defender's reward, respectively. Solid lines are four strategies in the information set. The ϵ_a, ϵ'_d and ϵ''_d represents the attacker's and defender's maximal utility errors if strategy ξ^* is removed.

Function DefenderError(·) Given candidate policy to be removed ξ^* and alternative policy ξ' that the attacker plays instead, function DefenderError$(\xi^*, \xi', \kappa_a(I))$ computes the upper bound on the defender's utility error on region where ξ' (after removing ξ^*) is undominated. The linear program is as follows:

$$\text{max: } \epsilon_d \tag{4.6a}$$

$$\text{s.t.: } (\forall \xi \in \kappa_a(I) \setminus \{\xi^*\}) :$$

$$\sum_{z \in I} U_a(\xi^*, z) p_z + \epsilon_a(\xi^*, I) \geq \sum_{z \in I} U_a(\xi, z) p_z - \epsilon_a(\xi, I) \tag{4.6b}$$

$$(\forall \xi \in \kappa_a(I) \setminus \{\xi^*, \xi'\}) :$$

$$\sum_{z \in I} U_a(\xi', z) p_z + \epsilon_a(\xi', I) \geq \sum_{z \in I} U_a(\xi, z) p_z - \epsilon_a(\xi, I) \tag{4.6c}$$

$$|\sum_{z \in I} R(\xi^*, z) p_z - \sum_{z \in I} R(\xi', z) p_z| + \epsilon_d(\xi', I) + \epsilon_d(\xi^*, I) \geq \epsilon_d \tag{4.6d}$$

$$(\forall z \in I) : 0 \leq p_z \leq 1 \tag{4.6e}$$

$$\sum_{g \in G} p_s = 1 \tag{4.6f}$$

where $R(\xi, z)$ denotes the attacker's reward if the attacker performs policy ξ in state z. It computes the defender's upper bound on the maximal utility difference between ξ^* and ξ', where ξ' becomes a new best response policy. Constraint (4.6b) restricts LP to the region, where ξ^* dominated, taking into account the possible policy errors ($\epsilon_a(\xi', I)$ and $\epsilon_a(\xi^*, I)$). Constraint (4.6c) further restricts LP to the region, where ξ' is a new undominated policy (yields higher utility than any alternative policy $\xi \in \kappa_a(I) \setminus \{\xi^*\}$). Constraint (4.6d) ensures that ϵ_d finds the maximal upper bound on the defender's reward difference between policy ξ^* and ξ', including the policy errors.

The full algorithm for computing the error-bounded SUS is the same as algorithm for computing the exact SUS $\kappa_a(I)$ (Algorithm 6), except that in line 14 ErrorBoundedSUS is called (Algorithm 8) with ϵ parameter instead of the ExactSUS (Algorithm 7).

Function ErrorBoundedSUS sequentially eliminates the policies from S, therefore, the players' final errors can sum to more than ϵ. Fortunately, we can control the total error by discontinuing the further policy eliminations from S, when the error reaches a maximal allowed bound. However, experiments showed that the accumulated errors are neglectable.

4.6.3 Solving Games

Exact and Error-Bounded Algorithm (MILPs)

We use MILP 2.19 formulation described in Section 2.3.5 to compute the SSE strategy profile. For each attacker's policy, the MILP formulation requires creating one float variable (slack $slack_{\sigma_f}$) and one binary variable ($r_f(\sigma_f)$). Since there are exponentially many attack policies for an attack graph, the MILP formulation would result in too many binary variables and solving MILP would be computationally very challenging. Therefore, we restrict the

attacker's strategies to exact or error-bounded SUS, which reduces the number of considered attack policies and speeds up the MILP computation. By allowing ϵ errors in SUS to both players we can solve games for up to medium sized network topologies. We call *error-bounded MILP* the MILP version, where function ErrorBoundedSUS is called to compute the attacker's policies in the information sets. In the remainder of this section, we present a collection of *heuristic* algorithms that find strategies close to SSE in polynomial time in the size of the game tree.

Solving Perfect Information Heuristic (PI)

The perfect information approach assumes, that the attacker can perfectly tell apart the states in the information sets. To model it, we can create an identical game tree to the original game, but changing the information set structure so that each state is in unique information set. This game can be solved using backward induction. After the Chance acts, for every defender's action we compute the attacker's optimal attack policy with algorithm described in Section 4.6.1 and choose the action for the defender that maximizes his expected utility. These games can be solved in polynomial time in the size of the game [Letchford and Conitzer, 2010].

Solving Zero-Sum Heuristic (ZS)

In zero-sum games, any Nash equilibrium yields the same utility to the leader as any strong Stackelberg equilibrium, thus, we can apply efficient algorithms for ZS games. First, we convert the original game to a zero-sum game using (ZS1)–(ZS4) rules, as described in Section 4.5.2. Then use single oracle (SO) algorithm, as described in Section 3, we solve the game. The advantage of the single oracle algorithm is that it does not require computing all the attacker's attack policies in advance, but computes and adds to the game the attacker's attack policies during the computation as needed. The single oracle algorithm requires $\mathrm{BR}_a(\delta_d)$, which returns the attacker's best-response strategy given the defender's strategy. We compute the attacker's optimal attack policy in each of the attacker's information sets separately. From the defender's strategy δ_d we compute the probability distribution of states in each information set, and then apply algorithm described in Section 4.6.2 to compute the attacker's optimal attack policies. The attacker's final strategy is an ensemble of the attack policies for all information sets.

Correlated Stackelberg Equilibrium (CSE)

In [Conitzer and Korzhyk, 2011] the authors present an LP that computes SSE of a normal-form game in polynomial time. The LP finds the probability distribution over the outcomes in the matrix with maximal utility for the defender under the condition that the attacker plays a best response. Here, we represent our game as a collection of normal-form games, one per attacker's information set, and solve it using this LP. The defender selects a probability distribution over the states in an information set and the attacker chooses an attack action in that information set. We can formulate one LP that finds strong Stackelberg equilibrium for each information set (which is translated into a normal-form game). We additionally

add a global constraint to obtain a realization plan for the defender, and add the objective function to maximize the defender's expected utility over all information sets. However, the solution of this LP does *not* result in the SSE of the original game, but in a correlated SSE, where the mediator sends signals to the follower what action should he play. The probability distribution over the terminal states allow the attacker to distinguish the states in the information sets via the Chance player actions, which can be interpreted as the follower receiving signals in the information sets.

For each of the attacker's information set $I \in \mathcal{I}_a$ we construct a normal-form game as follows. The defender chooses action $c \in \mathcal{A}_d$ that leads to a state in that information set and the attacker chooses an attack policy $a \in \hat{\kappa}_a(I)$. The outcomes in the matrix game coincide with the outcomes in the original game. Additionally, we restrict the sum of probabilities of the outcomes that originate from nature's action s to be equal to $\mathcal{C}(s)$. The LP formulation follows:

$$\max \sum_{z \in \mathcal{Z}} p_z u_d(s_z c_z a_z) \tag{4.7a}$$

$$\text{s.t. :} (\forall I \in \mathcal{I}_a, a_1, a_2 \in \hat{\kappa}_a(I)) : \tag{4.7b}$$

$$\sum_{h \in I} p_{z'=ha_1} u_a(s_h c_h a_1) \geq \sum_{h \in I} p_{z'=ha_1} u_a(s_h c_h a_2) \tag{4.7c}$$

$$\sum_{z \in \mathcal{Z}} p_z = 1 \tag{4.7d}$$

$$(\forall z \in \mathcal{Z}) : p_z \geq 0 \tag{4.7e}$$

$$(\forall s \in \mathcal{A}_c) : \mathcal{C}(s) = \sum_{c \in \mathcal{A}_d} \sum_{a \in \kappa_a(sc)} p_{z=sca} \tag{4.7f}$$

Here, the only variables are p_z. It finds probability distribution over terminal states that maximizes the defender's expected utility (objective). Constraint (4.7c) ensures that the attacker plays policy a_1, which yields higher expected utility than any other alternative a_2, therefore, best-responds. Because there is no constraint that forces the attacker to play pure strategy, the attacker can correlated his strategy with the correlation device that chooses outcome of a game $z \in L$ with probability p_z. $\kappa_a(sc)$ refers to $\kappa_a(I)$, such that history $sc \in I$.

Example 4.6.3. *An example follows to demonstrate our approach on the game in Figure 4.10b. Information sets I_1 and I_2 in the game in Figure 4.10a correspond to the two matrix games in Figure 4.10b. The defender chooses probability distribution over states in I_1 with actions c_1 and c_3 (similarly in I_2 with actions c_2 and c_4). The probabilities of the terminal states of the game tree correspond to the probabilities of the outcomes in the matrix games (p_1 through p_8). Additionally, the probabilities of outcomes p_1, p_2, p_3, p_4 must sum to $\mathcal{C}(s_1)$, since they root from the same action s_1 of the Chance player (the same case for s_2).*

This LP has weaker restrictions on the solution compared to the MILP formulation for SSE [Bošanský and Čermák, 2015], since it does not restrict the attacker to play a pure best response. Similarly to SSE, it maximizes the defender's expected utility. Therefore,

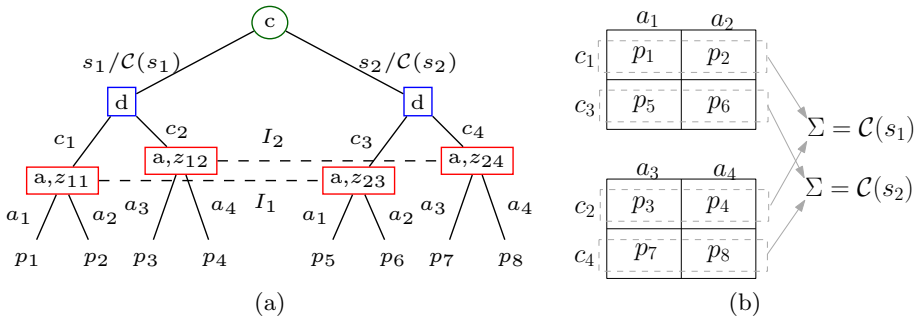


Figure 4.10: The extensive-form game in (a) translated into two normal-form games in (b).

the defender can obtain the same or higher utility than in SSE. Formulating game with a CSE has a similar drawback as the MILP formulation, namely, that it requires computing SUS for each of the attacker's information set in advance of the game solving.

4.7 Experiments

In this section, we experimentally evaluate the proposed algorithms in terms of runtime and the quality of the found strategies. The structure of this section is as follows: in Section 4.7.1 we describe the individual networks and attack graphs used in the experiments, in Section 4.7.2 we compare proposed algorithm to compute the optimal attack policy with the domain-independent solvers; in Section 4.7.3 we compare the runtimes of the exact, error-bounded and heuristic algorithms for SSE and the quality of the computed solutions, in Section 4.7.4 we answer several problem-related security questions, and in Section 4.7.5 we compare the game theoretic strategies against the human behavior.

4.7.1 Experimental Setting

We use five different computer network topologies, as depicted in Figure 4.11, with varying complexities to generate the attack graphs for the games. Labels $(remote, local, client)$ at the hosts denote the number of remote, local and client vulnerabilities of that host type in an attack graph.

We use MulVAL to generate the attack graph structures for these networks. Unfortunately, MulVAL provides very limited spectrum of the action probabilities, usually with only two or three different values, while the action costs are not provided at all. To simulate more realistic cases with various action costs and probabilities, we decided to draw the probabilities and costs from the normal distributions. The action costs we draw (in dollars) from the normal distribution with mean 50 and standard deviation 50 (in dollars), which assumes that the exploits are not too expensive, as they are often available online (e.g., in Metasploit[2]). On the other hand, attempting to perform them is not completely for free, as the attacker still spends some time by installing the tools and executing the exploits. The

[2]www.metasploit.com

action success probabilities we draw from the normal distribution with mean 0.7 and standard deviation 0.2 with interval bounds $[0, 1]$. We experimentally chose these values, where the probabilities are not too low to demotivate the attacker from attacking immediately and not too high to allow the attacker's to easily always compromise the networks.

For each network, as depicted in Figure 4.11, we generate the Chance's actions $\mathcal{A}_c$ as follows. Each network consists of exactly *one* host of a *black* host type by default. The Chance player extends the default network by adding up to N hosts from T host types, which includes black and *gray* host types. After the Chance player, there are $|\mathcal{A}_c| = \sum_{i=0}^{N} |T|^i$ networks for the defender to configure.

For each network, the defender similarly chooses up to k honeypots to deploy from T host types, resulting in $|\mathcal{A}_d| = \sum_{i=0}^{k} |T|^i$ different possible honeypot allocations. Increasing the parameter T, increases the complexity of the network attack graphs, the number of Chance's and the defender's actions. Next, each network topologies is explained in detail.

Business network is inspired by a network in [Homer et al., 2013], and resembles a smaller company computer network. For $T = 3, 4, \ldots$, we increase the network complexity by adding host types VPN, Firewall, PC_2, PC_3, etc., which results in attack graphs with 7, 13, 16, 22 attack actions, in this order.

Chain network consists of PC types in a grid-shaped topology with two rows and a variable number of columns. We increase the network's complexity by adding a column of PC's, where all hosts in the last column have direct visibility to the database. For $T = 3, 4, \ldots$, the attack graphs consist of 7, 15, 23, 31 attack actions, in this order.

LocalPlus is a simple network topology. For $T = 4, 5, \ldots$, the attack graphs consist of 8, 11, 14, 17 attack actions, in this order.

TV network is inspired by a network used during cyber security exercises in Swedish Defence Research Agency in 2012 ([Sommestad and Sandström, 2015]), where the networks were deliberately left vulnerable to the attacks. For $T = 5, 6, \ldots$, the attack graphs consist of 26, 29, 32 attack actions, in this order.

Unstructured (unstr) is a synthetic network consisting of host types that are directly visible to the attacker from the Internet. Because the atomic actions are not dependent on each other, these attack graphs provide the largest number of possible attack policies with all possible action combinations. For $T = 1, 2, \ldots$, the attack graphs consist of 3, 6, 9, 12 attack actions, in this order.

In all networks, the values of the host types are set as follows: $r_{Database} = r_{Studio} = 5000, r_{Server} = 2500, r_{PC} = r_{VPN} = 500$. The honeypot cost factor is set to $\gamma_c = 0.02$, which corresponds to 2% of the host type purchasing cost.

4.7.2 Optimal Attack Planning

In this section, we evaluate the effectiveness of the pruning techniques that we use for the computation of the optimal attack policies, as described in Section 4.6.1. We compare our MDP approach to two domain-independent MDP solvers.

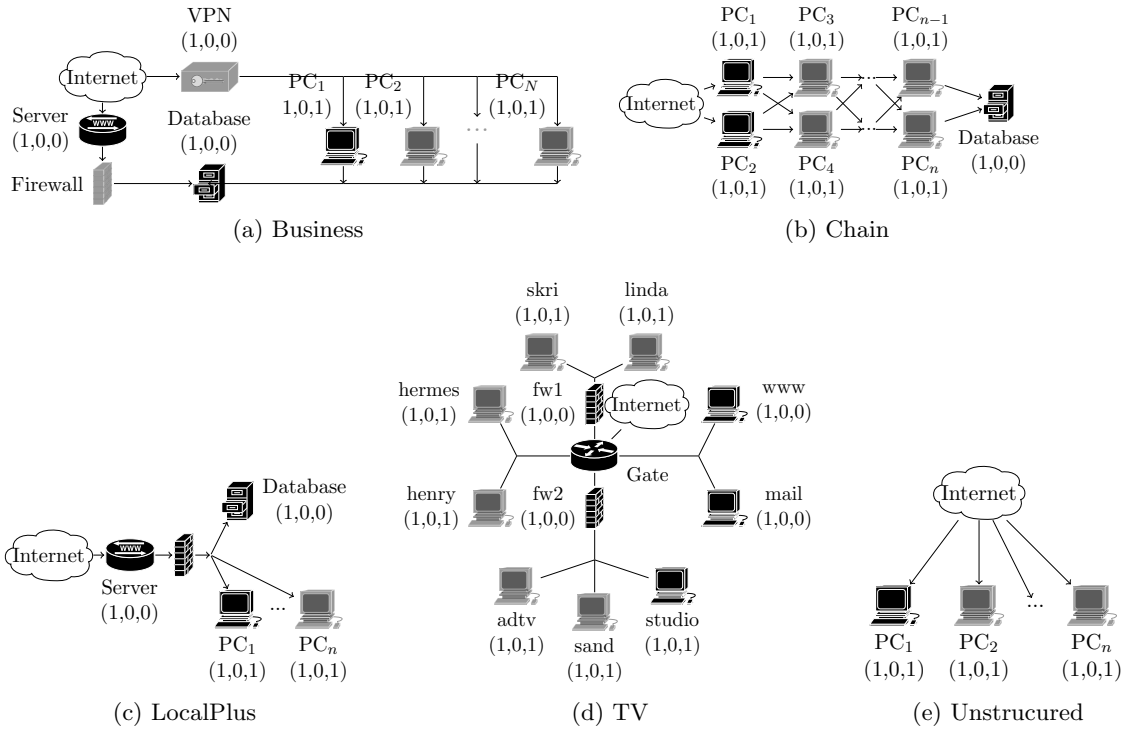


Figure 4.11: Networks used in the experiments

Pruning Techniques Evaluation

We evaluate the contribution of the Sibling Theorem (ST), and for branch and bound technique the lower bound (LB) and upper bound (UB) separately. In all evaluations, we use dynamic programming (caching). Since techniques may prune the same branches, we measure each technique's contribution by measuring the algorithms slowdown after disabling the pruning technique. Thus, we compare 5 settings: *none* (no techniques is used), *all* (all techniques are used), *-ST* (all but ST), *-LB* (all but LB) and *-UB* (all but UB). For each setting in Figure 4.12 we show the optimal attack policy computation runtimes for different sizes of Business and Chain networks. In Business network, adding vulnerabilities to a workstation creates large decision points. Since all added vulnerabilities have the same effect, they belong to the same OR-sibling class and are effectively pruned by the sibling-theorem. Therefore, the optimal action is selected without additionally extensive computation (compare *all* to *-ST*). In Chain network the branch and bound helps the most (*-LB* and *-UB*). This experiments were ran for LocalPlus networks also, with similar results as for Chain network, thus, we omit the results for LocalPlus. In all experiments, we apply all techniques since they do not seem to have any negative impact and can dramatically speed up the computation.

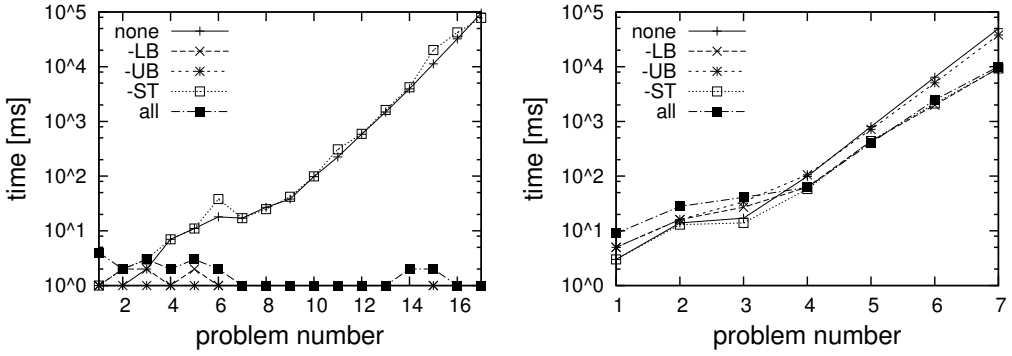
Table 4.1: (a) Computation times (in seconds) of computing optimal attack policies by DP, GUIDO and SPUDD. (OoM) - Out of Memory, (Err) - planner exited with segmentation error.

Problem	# Actions	Our approach	UID	SPUDD
LocalPlus-3	16	**<1**	71	1
LocalPlus-6	28	**3**	(OoM)	125
LocalPlus-8	36	**406**	(OoM)	1951
Chain-3	20	**<1**	21,68	7
Chain-5	32	**<1**	(OoM)	2646
Chain-7	44	**<1**	(OoM)	Err

Comparison with Other MDP Algorithms

We compare our algorithm for computing the optimal attack policies on the attack graphs to other two approaches. The first approach converts an attack graph to an *Unconstrained Influence Diagram* (UID) as described in [Lisý and Píbil, 2013] and uses GUIDO [Isa et al., 2007] to solve it. The second approach translates attack graph to probabilistic planning problem, which can be solved using SPUDD solver ([Hoey et al., 1999]), which iteratively improves the policy until it is optimal. SPUDD solver won 3rd place at the International Planning Competition (IPC) in the MDP track in 2011. We chose SPUDD because it guarantees to return an optimal contingency policy, while the other two winning planners from IPC do not.

In Table 4.1, we present the runtimes of each algorithm. Our algorithm dramatically outperforms the other two approaches, where the hardest network LocalPlus-8 was solved 5x faster than by SPUDD planner. For larger problems, the UID solver exceeded the limit of 8GB and the computation was terminated.



(a) Business network (b) Chain network

Figure 4.12: Time comparison of the pruning techniques. Legend: none - no pruning technique, (-LB) - all but lower bound, (-UB) - all but upper bound, (-ST) - all but Sibling Theorem, (all) - all pruning techniques.

4.7.3 Game Theoretic Algorithms Comparison

Scalability of the Algorithms

Here, we analyze the size of the problems that each game-theoretic algorithm is able to solve. For each network, we ran three instances with different random seeds to draw the action success probabilities and the costs in the attack graphs. We increase the problem complexity by increasing parameters T, N, and k. For each experiment we set 2 hour time limit.

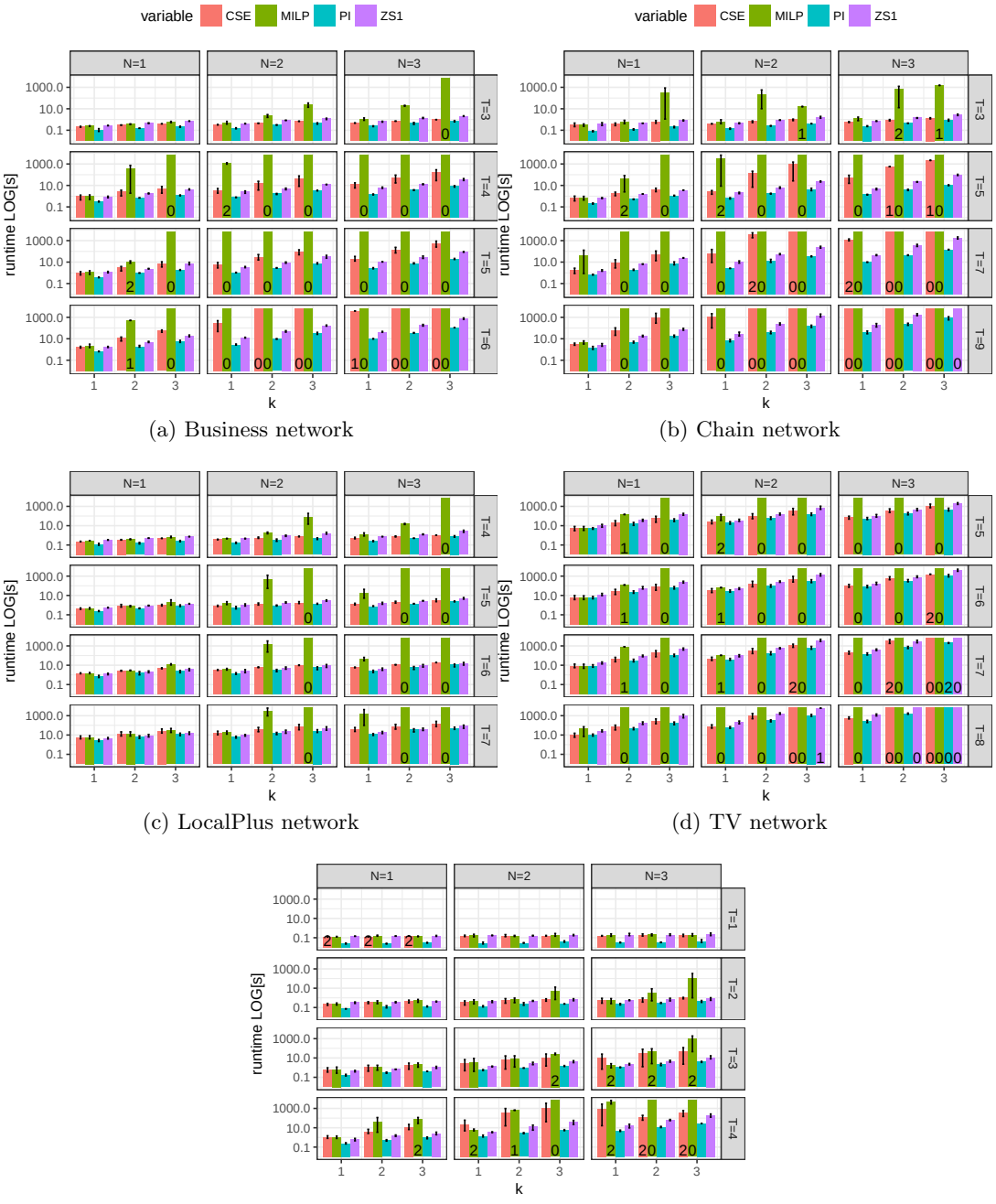
In Figure 4.13 we show the average, minimal and maximal runtime (as error bars) for each algorithm and network type that finished within the given time limit. The number on each bar denotes the number of finished instances (if less than 3). The results indicate that computation of the error-bounded strategies scales worse than the heuristic approaches. It found a solution in only 49% of the problems within the 2-hour time limit. CSE was able to solve larger instances than the MILP, and 88% of the problems in total. Both algorithms, first, have to compute SUS for each information set in the game before engaging in the computation of the solution concepts.

All zero-sum approaches have similar scalability, so we present only ZS1. The zero-sum and PI approaches do not require constructing SUSs in advance, so they were able to solve 97% and 99% of the problems, respectively. We were able to reach their limits in the game with TV network topology ($T = 8, n = 3, k = 3$) which consists of $|\mathcal{A}_c| = |\mathcal{A}_d| = 585$ actions for the Chance player and the attacker and 3552 information sets for the attacker. In all instances, PI was faster than zero-sum approach.

Solution Quality

In this section, we analyze the quality of the strategies computed by the heuristic approaches and compare them to the error-bounded MILP. We use the concept of *relative regret* to capture the relative difference in the defender's utilities for using one strategy instead of another. The relative regret for playing strategy δ_d instead of δ_d^* is $\rho(\delta_d, \delta_d^*) = 100 \cdot \frac{u_d(\delta_d^*, BR_a(\delta_d^*)) - u_d(\delta_d, BR_a(\delta_d))}{|u_d(\delta_d^*, BR_a(\delta_d^*))|}$ [in %]. It captures the defender's relative loss assuming that the attacker responds with the best-response against both strategies. The higher the regret $\rho(\delta_d, \delta_d^*)$ the worse strategy δ_d is compared to strategy δ_d^* for the defender. In the case the defender's strategy yields zero utility we do not add this instance in the total average due to the division with zero. It can happen, when the attacker's actions have low success probabilities and high action costs, so the attacker does not attack even if no honeypots are deployed. This happened only for unstructured network topology in 9 out of 144 instances. Additionally to the heuristic game-theoretic approaches, we examined the quality of three baseline strategies.

Baseline Strategies *Uniform* (Uni) strategy deploys all honeypots as any host type with an equal probability. This strategy produces high uncertainty for the attacker in the information sets. However, it does not consider that the attacker focuses primarily on attacking the valuable host types. The second, *most-rewarded* (Rew) baseline approach, deploys only honeypots of the most valuable host type. While it may protect the most valuable host type, the attacker can confidently attack the remaining unprotected host



(a) Business network

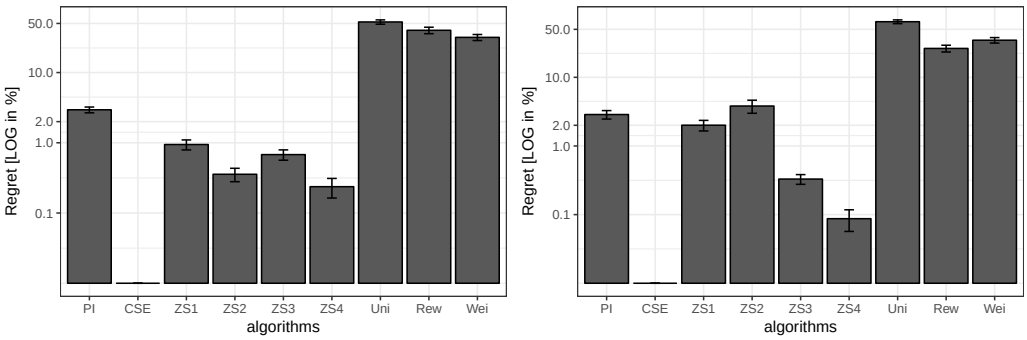
(b) Chain network

(c) LocalPlus network

(d) TV network

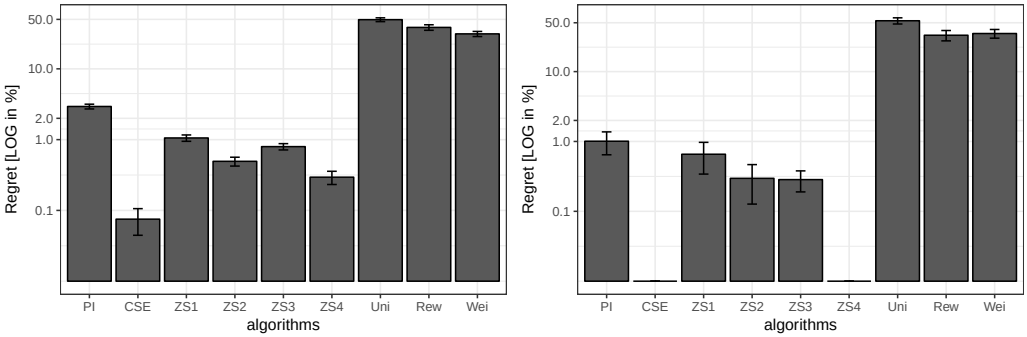
(e) Unstructured network

Figure 4.13: Average, minimal and maximal (the error bars) runtimes in seconds for each algorithm, network topology, number of host types T, parameter to generate networks for the Chance player N and number of honeypots k.



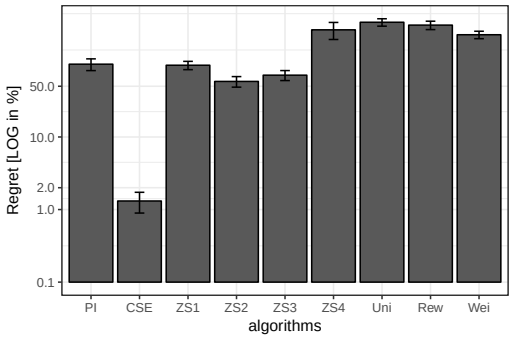
(a) Business network (37 instances).

(b) Chain network (32 instances).



(c) LocalPlus network (60 instances).

(d) TV network (16 instances).



(e) Unstructured network (74 instances).

Figure 4.14: The defender's average relative regret and standard error (error bars) for the strategies computed by the heuristic algorithms and the baseline approaches for each network

types. The last baseline is *weighted-reward* (Wei), which is a mixture of the previous two baseline methods. In each network, it deploys a honeypot of type t with the probability proportional to the value of that host type. Formally, for type t the probability is $\frac{r^t}{\sum_{t \in T} r^t}$. This is a reasonable assumption that the defender protects host types proportionally to

their values, however, this (and other baseline methods) completely neglects the attack graph structure.

In Figure 4.14, we present a comparison of the defender's utility regrets if the heuristic strategies are used instead of the error-bounded MILP (notice logarithmic y-axis). We compute the regrets only for the settings, where all the algorithms found strategies within the 2 hour limit (including MILP). The result shows that CSE found strategies with the lowest overall regrets and in 88% of the problems it computed a strategy with zero regret, which is optimal with regards to the error-bounded MILP.

Concerning the zero-sum heuristics, ZS4 performs the best in all networks except the unstructured network. In the ZS4 approach the defender's utility is augmented to prefer outcomes with expensive attack policies for the attacker. Therefore, the ZS4 approach works well for networks where long attack policies are produced (e.g., TV or Chain networks) but worse in the cases, where the attack policies are cheap, such as the unstructured networks. PI heuristic had relative regret of 2% for all the networks except the unstructured network again. This heuristic can be useful for solving the problems, where the algorithms with lower regrets are intractable since this algorithm is the fastest. The defense strategies for the problems with unstructured network topology have higher regrets compared to the other networks because there is no such highly-valued host type as are in the other networks. The defender's utilities in the unstructured networks are closer to zero, thus smaller changes in the utility result in higher regrets.

Although ZS4 heuristic performed the best among the zero-sum heuristics, each of these heuristics present certain drawbacks that we would like to discuss. In the ZS1 and ZS2 the defender ignores his costs for deploying the honeypots; these strategies often deploy the honeypot of database type, which is the most expensive action. For example, the ZS2 strategies deploy a database in 74% of the problems, while the CSE only in 43% of the problems. Strategies computed by the ZS3 and ZS4 are more difficult to analyze. They often miss the precise probability distribution of the states in the information set that makes the attacker indifferent between two or more attack policies, in which case the attacker does not always choose the policy in favor of the defender as it is expected in SSE. However, we were not able to find a general error that the zero-sum heuristic strategies make.

The baseline strategies, that do not consider the attacker's possible reactions to the defense strategies, resulted on average relative regret about 56%. Even the worst game theoretic algorithm (PI) performed better than any baseline solution. This shows that game-theory is essential and should be considered while security defense strategies are developed.

Finally, let us present the player's utility errors for using error-bounded SUS in the MILP and CSE algorithms. For the error-bounded SUS algorithm we set $\epsilon = 10^{-5}$ and for all experiments gathered the accumulated maximal errors that both players may experience. The maximal relative error for the defender and the attacker across all experiments is only $5.4 \times 10^{-3}\%$ and $1.6 \times 10^{-2}\%$, respectively. In practice, such small relative regrets can be neglected, since other estimates, such as game utilities or the Chance probabilities may result in higher regrets for the defender than this. In some information sets in problems with TV network, the error-bounded SUS reduced the number of policies from 500 to 4, which significantly decreased its runtime.

4.7.4 Essential Network Security and Honeypot Related Analysis

The presented game-theoretic framework can be used to answer interesting questions. For instance, what is the optimal number of the honeypots to deploy into the network, what is the defender's regret for not using the honeypots, or for not exploiting the attacker's imperfect information in the game? In this section, we answer some of the representative questions using our game-theoretic framework to demonstrate the potential usefulness of this work and game theory in general.

Sensitivity Analysis

The defender's optimal strategy highly depends on the attack graph structure, the action costs, success probabilities, and rewards. In real-world scenarios, the defender can only estimate these values. Since the ZS4 approach has a good trade-off between its scalability and the solution quality, we analyze the sensitivity of the defender's strategies computed with the ZS4 to perturbations in action costs, success probabilities, and values of host types in attack graphs.

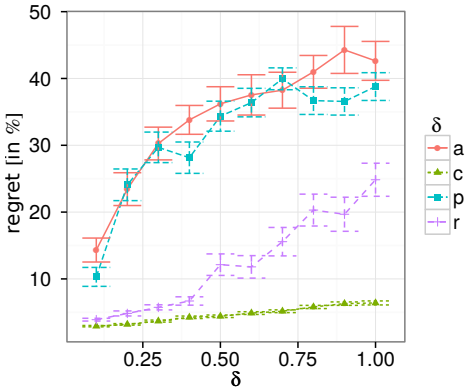


Figure 4.15: The defender's regret for perturbed action success probabilities, action costs, and host type values

We generate the defender's attack graph estimation by perturbing the original attack graph actions as follows: (i) action success probability are chosen uniformly from the interval $[p_a - \delta_p, p_a + \delta_p]$ restricted to $[0.05, 0.95]$ to prevent it becoming impossible or infallible, (ii) action costs are chosen uniformly from interval $[c_a(1-\delta_c), c_a(1+\delta_c)]$, and (iii) rewards for host type t uniformly from the interval $[r_t(1-\delta_r), r_t(1+\delta_r)]$, where $\delta_p, \delta_c, \delta_r \in [0,1]$ are perturbation parameters. The action probabilities are perturbed absolutely (by $\pm\delta_p$), but the costs and rewards are perturbed relative to their original value (by $\pm\delta_c c_a$ and $\pm\delta_r r_f$). The intuition behind this is that the higher the cost or reward values, the larger the errors the defender could have made while estimating them, which cannot be assumed for the probabilities.

We compute (i) the defender's strategy δ_d on the game with the original attack graphs and (ii) the defender's strategy δ_d^p on the game with the perturbed attack graph. In Figure 4.15 we present the mean and standard errors of the defender's relative regret for using perturbed (estimated) attack graph instead of the attacker's (real) attack graph. In both cases, the attacker best-responds according to the attacker's attack graph. The relative regret is computed as follows: $100 \cdot \frac{u_d(\delta_d, BR_a(\delta_d)) - u_d(\delta_d^p, BR_a(\delta_d^p))}{|u_d(\delta_d, BR_a(\delta_d))|}$ [in %], where $\mathrm{BR}_a()$ are computed on non-perturbed attack graphs.

For instance, 20% error in the action probabilities ($\delta_p = 0.2$) results in a strategy with 25% lower expected utility for the defender than the strategy computed based on the true values. However, the same 20% error in the action costs or host type rewards results in only

Table 4.2: The defender's relative utility regret [in %] for not using up to k in the network.

	Network				
k	business	chain	localPlus	TV	unstructured
1	52.9	87.6	49.4	90.5	682.64
2	117.8	170.8	120.4	217.7	1083.15
3	184.54	233.3	185.8	364.2	1183.94

5% lower utility. According to the results, the regret depends significantly on the action success probabilities, then on the rewards, and depends least on the attacker's action costs. This is an important result for consideration of the future direction of this work. It suggests, that estimating the action probabilities should be done as precisely as possible.

Honeypot Contribution to Network Security

It is important to quantitatively measure *how honeypots contribute to securing the networks* Organization often require some estimates before decision is made whether or not to deploy the honeypots. In Table 4.2, we present the defender's average relative utility regret for not deploying any honeypot instead of deploying strategically up to k honeypots according to the error-bounded MILP. For example, if the defender uses no honeypots for business network, the attacker can harm the defender on average by 184.54% more compared to the case that the defender uses up to 3 honeypots.

The regrets differ across the network, since some of them are initially harder to exploit, e.g., the Chain network has Database located far from the Internet and the attacker needs to compromise many intermediate host types. Additionally, notice an interesting observation, that for all networks (except the unstructured) the regret for not deploying two honeypots ($k = 2$) is larger than twice the regret for not using one honeypot ($k = 1$). Stated differently, the defender can intelligently allocate two honeypots to produce even higher utility benefit than the sum of the benefits from the individual honeypots. For some networks, the same holds for three honeypots, but the benefit of each additional honeypot is expected to decrease, as we will see later in this section. Knowing how much the honeypots contribute to the network security may play an essential role for the companies future security decisions.

Regret for PI

In [Durkota et al., 2015b] it was shown that in games where the attacker has perfect information (PI) about the network, the defender either (i) duplicates the most valuable host type, or (ii) deploys honeypots of the host types in the "choke-points" of the network, which is a set of hosts that the attacker must always pass in order to compromise a target or a valuable host. In games where the attacker has imperfect information about the networks, the defender's decision-making is more complicated, as he must weigh whether to adopt one of the previously mentioned PI strategies or to exploit the attacker's imperfect information in the information sets.

In Table 4.3 we present the defender's regret for playing the perfect information strategy instead of the MILP strategy (these values can also be seen in Figure 4.14). The average relative regret is only 2.5% to 3%, which suggests that the attacker's uncertainty can be exploited to the defender's benefit, but not by much. This is a very important result, showing that we *can* use a fast PI heuristic to compute the defense strategy at the cost of relatively low utility loss.

Attacker's Uncertainty in Information Sets

Let us examine *how* the MILP strategy exploits the attacker's imperfect information in the game. We aim to understand the degree of the uncertainty that the attacker faces in the information sets due to the defense strategies. We use widely known *Entropy* to measure the unpredictability of each state in the information set for the attacker. In Table 4.4 we show the weighted average entropy across all of the attacker's information sets computed by MILP and PI strategies. The higher the entropy is, the less predictable states are for the attacker in the information sets. Although the PI heuristic strategy does not explicitly exploit the attacker's imperfect information, in some cases it may play to the same information set by chance and create uncertainty for the attacker, as can be seen in the table. Thus, the entropy for the PI can be considered as a baseline value.

The entropy measure is not easy to interpret in terms of state probabilities, nevertheless, we can see that the MILP strategy has twice as much entropy as the PI strategy, thus, the attacker faces higher uncertainty in the information sets. MILP is more active in exploiting the attacker's imperfect information than PI, which increases the defender's expected utility by the above-mentioned 1–3%.

Incorrectly Assuming Imperfect Information

Let us now consider what happens if the imperfect information assumption, assumed and exploited by the MILP strategy, is not met. This may result in high utility loss for the defender. In Table 4.5, we show the defender's relative utility loss for assuming (and playing optimally against) that the attacker has imperfect information, while the attacker has perfect-information according to which he best-responds. The defender's relative utility loss for playing strategy δ_d and the attacker switching from δ_a to δ'_a we compute as $100 \times \frac{u_d(\delta_d,\delta_a)-u_d(\delta_d,\delta'_a)}{u_d(\delta_d,\delta_a)}$ [in %].

For Business, Chain and Local network the regret can be as high as 9%, which is almost three times as large as the defender's relative regret for playing PI strategy instead of MILP (see Table 4.3). The perfect-information attacker can exploit the defender's attempt to exploit the attacker's incorrectly assumed imperfect information. In fact, considering the

Table 4.3: For each network topology we show the defender's relative regret for playing PI instead of MILP.

	unstructured	business	chain	local	TV
MILP vs PI	100.69 ± 18.53	2.95 ± 0.28	2.87 ± 0.40	2.94 ± 0.23	1.01 ± 0.36

Table 4.4: For each network topology we show the weighted average Entropy for the attacker in the information sets due to MILP and PI strategies.

	unstructured	business	chain	local	TV
Entropy for MILP	0.45 ± 0.1	0.48 ± 0.1	0.44 ± 0.1	0.51 ± 0.1	0.32 ± 0.1
Entropy for PI	0.02 ± 0.1	0.26 ± 0.1	0.28 ± 0.1	0.28 ± 0.1	0.27 ± 0.1

results, the defender should use MILP strategy *only* in the case when the attacker believes that the attacker has imperfect information with likelihood higher than 1/3. Otherwise, it is better to play a safe PI strategy, which does not attempt to exploit the attacker's imperfect information and at the same time does not put the defender at risk.

Defender's Utility Cost Components

Companies are often interested in the individual cost components of the solutions. Here, we analyze the defender's different cost components ($-R$ and $-C_d$) that the defense strategy yields. In Figure 4.16 we present the defender's cost for having compromised the network (solid lines) and the expected cost for the deployed honeypots (dashed lines) for different honeypot cost factors. We scale the number of the available honeypots for the defender ($k = 1, \ldots, 15$) with two different honeypot deployment cost factors, cheap honeypots ($\gamma_c = 0.02$) and expensive honeypots ($\gamma_c = 0.1$). We choose ZS4 for its scalability and low relative utility regret for the defender. With 0 honeypots, the defender receives utility -3000 for having a vulnerable network and 0 for the honeypot costs. With only 1 honeypot, the utility for the network increases to -2000 at relatively low cost for the honeypots. With $\gamma_c = 0.02$ and $k = 12$ we can see that the network is almost secure and most expenses are spent due honeypot costs. This analyses allows estimating the return value of the investment into the honeypots.

Optimal Number of Honeypots

Another natural question that arises is how many honeypots the defender should deploy, taking into account the honeypot costs, to maximize the defenders expected utility. To find the optimal number of honeypots, we increase the maximum number of allowed honeypots k and for each find ZS4 strategy. In Figure 4.16, we show the average number of honeypots that the defender deploys in each of the networks after the Chance player. For the honeypot cost factor $\gamma_c = 0.1$ the defender uses 4.6 honeypots on average, while with cheaper honeypots ($\gamma_c = 0.02$) the defender deploys 11.5 honeypots on average. Deploying

Table 4.5: The defender's relative loss if the defender assumes and plays optimally against the attack with imperfect information, while the attacker best-responds with a perfect information.

	unstructured	business	chain	local	TV
MILP vs MILP-PI	515.21 ± 73.6	9.07 ± 0.5	9.46 ± 1.2	9.08 ± 0.4	2.84 ± 1.1

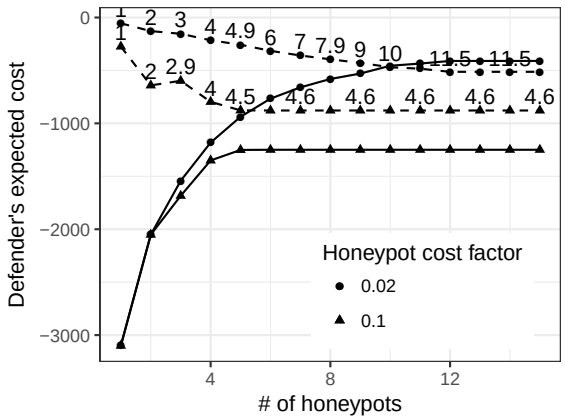


Figure 4.16: The defender's cost for having his network compromised $R_d(\delta_d, \delta_a)$ (solid line) and cost for deploying honeypots $C_d(\delta_d)$ (dashed line) for different number of honeypots k in business network with $N = 1, T = 3$. For each k we depict the expected number of honeypots (values) used in each network for different honeypot cost factors.

any additional honeypot costs the defender more than the honeypot contributes to securing the network. Converging to a specific number of honeypots does not necessarily imply that with $k + 1$ honeypots the defender's strategy would not deploy $k + 1$ honeypots, but it is unlikely.

From business perspective, this shows that if the honeypot cost drops from $\gamma_c = 0.1$ to $\gamma_c = 0.02$ (by 80% cheaper), the defender would purchase 12 honeypots instead of 5.

Choice of Honeypot Types

Finally, notice in Figure 4.16 that for $\gamma_c = 0.1$ and $k = 2$ the honeypot costs are higher than with an additional honeypot ($k = 3$). With the third honeypot the probability of deploying the cheaper server honeypot has increased from 0.51 to 0.73, while the probability of deploying the twice more expensive database honeypot has decreased from 0.37 to 0.02. With three honeypots, the defender has lower honeypot deployment cost (dashed line) *and* secures better the networks (solid line). The defender is able to deploy cheaper honeypots in more strategic locations (e.g., network bottlenecks). Therefore, deciding which honeypot types to deploy should be thought through in advance and strategically, as for different k the deployed honeypot types may be very different. (In practice it means that one cannot purchase and optimally allocate n honeypots first, and later purchase an additional honeypot and allocate them optimally; since the optimal strategy with $n+1$ honeypot may require completely different honeypot types than the strategy for n honeypots.)

4.7.5 Case Study

We conducted a survey to compare the performance of the game-theoretic solutions to human opponents. The 45 respondents were participants in a four-day-long forensic malware

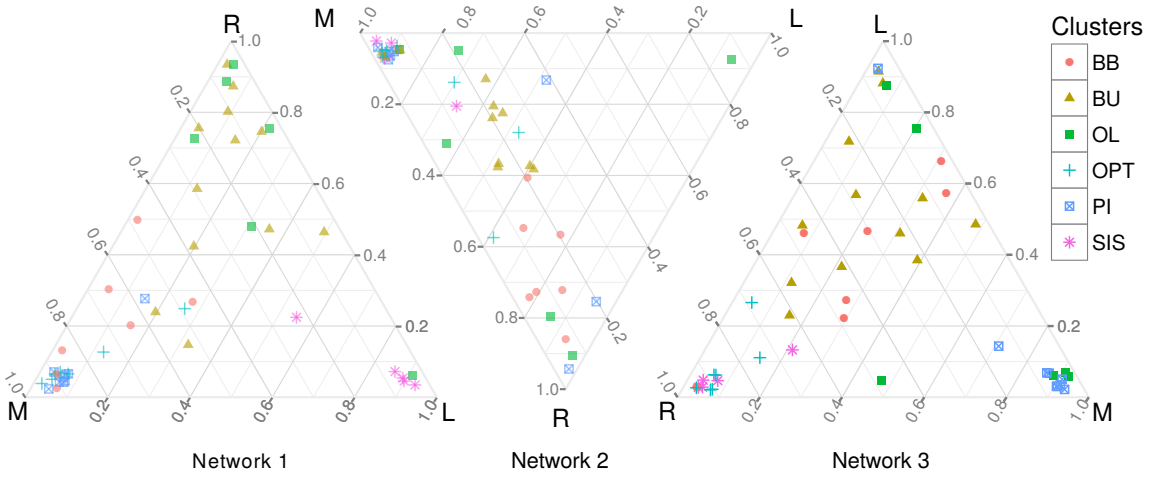


Figure 4.17: Respondent's defense strategies and a clustering of the strategies. Each triangle represents the space of possible defender strategies for the network in Figure 3.1 (each corner is a pure strategy).

Table 4.6: Defender's utility, standard deviation, 10th and 90th percentile for the defender's strategies (rows) and attacker's strategies (columns). BR_a - attacker's best-response strategy to defense strategy; MR_d,MR_a - mean respondent defense (resp. attack) strategy.

Defense / Attack	Best Response BR_a			Mean Respondent MR_a		
	mean	std	10th, 90th percentile	mean	std	10th, 90th percentile
Optimal (OPT_d)	-207	131	-350, -50	-189	181	-350, 100
Respondent (R_d)	-285	236	-500, 100	-167	208	-360, 100
Mean Respondent (MR_d)	-262	266	-500, 100	-164	207	-360, 100
Baseline uniform	-317	322	-800, 100	-162	240	-500, 100
Best Response (BR_d)	-302	187	-500, -50	-111	200	-500, 100

seminar, members of multi-agent technology group at CTU, and employees of two computer security companies. The survey was presented as a competition among the respondents to motivate the respondents to create effective strategies.

To avoid overwhelming participants with too much information, we used the simple networks in Figure 4.4 and set all honeypot costs and attack action costs to zero. We kept the reward of 1,000 for target host T and added penalty of 200 for the attacker if detected (as a part of the attacker's cost), and the exploit success probabilities of 0.2 and 0.8 for host-types A and B, respectively. Attacking routers and host-type T is always successful, but the attacker can initiate attack only from host-types A or B.

4.7.6 Respondent Behaviors

Analysis of 45 collected responses revealed five main clusters in the respondents' defense strategies. We compared this clustering to clustering of 500 uniformly random data sets using five standard quality metrics. The quality of our clustering was better than the 95th

percentile indicating that the clusters are not likely to appear by chance. A strategy belongs to a cluster if its L1 distance from a hand-selected centroid strategy is less than 1/3.

For each cluster we present the percentage of the strategies in that cluster and defender's average expected utility (u_d) of the strategies against a worst-case attacker. The clusters can be described as follows:

1. **Optimal Strategy ($\mathbf{OPT}_d$)** ($15\%, u_d = -236 \pm 23$) is a cluster around the game theoretic solution.
2. **Perfect Information ($\mathbf{PI}_d$)** ($26\%, u_d = -267 \pm 21$) defends each network in isolation by playing into I1, I4 and I5, not exploiting the attacker's uncertainty.
3. **Single Information Set ($\mathbf{SIS}_d$)** ($11\%, u_d = -343 \pm 16$) in contrast to PI_d plays always to I5.
4. **Biased Uniform ($\mathbf{BU}_d$)** ($26\%, u_d = -292 \pm 22$) mostly allocates one honeypot to each side, and with probabilities 0.1 and 0.2 both honeypots to A and B, respectively.
5. **Both to B ($\mathbf{BB}_d$)** ($15\%, u_d = -255 \pm 24$) protects more vulnerable host-type B always with two honeypots.
6. **Outliers ($\mathbf{OL}_d$)** ($13\%, u_d = -334 \pm 111$) are defense strategies with larger distance than 1/3 to any centroid strategy.

We visualize the individual defense strategies in Figure 4.17. Each triangle represents one possible network and the points represent the normalized probability distributions over pure strategies (each corner represents a pure strategy).

4.7.7 Survey Analysis

In Table 4.6 is a summary of the strategy analysis. Each entry contains the defender's mean utility, standard deviation (std), 10th and 90th percentile from 10^6 simulations for a pair of a defense strategy (row) against an attack strategy (column).

The OPT_d strategy maximizes the defender's utility against worst-case attacker. Column BR_a is best-response (worst-case) attack strategy against each defense strategy, which shows how *exploitable* a defense strategy is. The R_d represents the respondents' defense strategies in our dataset. We generated a single "mean" respondent strategy by averaging the individual strategies, labeled MR_d. By comparing these strategies to the baseline uniform strategy, we see that respondents played better than the random baseline.

The respondents' mean attack strategy MR_a reveals some interesting observations. MR_d against MR_a has lower mean utility than the optimal strategy OPT_d against MR_a. It means that against the human attackers, it might be better to defend with human defense strategy MR_d rather than OPT_d. However, MR_d may not be a good strategy in the long-term perspective. Attackers will likely learn from experience and move towards the best-response BR_a by increasing the frequency of successful attacks and reducing failed ones [Kalai and Lehrer, 1993]. Instead of playing OPT_d, it might be better to begin with MR_d strategy, but switch to OPT_d as the attackers start adapting.

However, we can develop even better defense strategy than OPT_d or MR_d, which exploits the human behavior. BR_d is the defender's best-response defense strategy against the human MR_d attack strategy. This results in highest utility for the defender; however, there

is added risk because it is vulnerable against BR_a attackers who specifically exploit this strategy.

Our strategies have large standard deviation due to two factors. First, networks have different levels of security (Network 3 is more vulnerable than Network 1) and more vulnerable networks will naturally have lower utility. For example, with strategy OPT_d the administrator of Network 1, 2, and 3 has expected utility -23, -247, and -350, respectively. Second, randomized strategies are necessary to minimize the strategy's predictability, but they also result in variability in outcomes. The OPT_d strategy has the lowest standard deviation against both types of attackers, which can be a desirable property.

4.8 Conclusion

In this chapter, we demonstrate that it is possible to use game theory to model complex contingency plans for an attacker using attack graphs, and integrate these into a game-theoretic analysis of how the defender should optimize security resources. We can model the realistic uncertainty that an attacker may have about the network structure, and incorporate this into the analysis of the optimal defensive policy. We focused on a set of deception-based strategies for defenders based on deploying honeypots in a computer network. Finding the optimal deployment when considering both the complexity of possible attack plans and uncertainty about network structure is computationally challenging, but we were able to find effective algorithmic strategies for solving these problems by considering approximation and heuristic strategies.

Our contribution is the attacker's behavior model using the concise attack graph representation and the algorithms that compute the attacker's optimal attack policies using (PO)MDP models. We developed and experimentally evaluated the effective pruning techniques to *avoid* building the full (PO)MDP state search space. These algorithms are used as a subroutine for game-theoretic algorithms that compute players' SSE strategy profile.

Our second main contribution is the game model and set of algorithms to solve them optimally and suboptimally. By introducing a neglectable error to both players, we were able to significantly reduce the size of the game and compute the error-bounded strong Stackelberg equilibrium strategies for small and medium-sized problems. In addition, we introduced and evaluated several heuristic approaches based on more restrictive assumptions, e.g., that the attacker has perfect information, or that the game has zero-sum utilities.

One of the most interesting approaches, CSE reaches greater scalability than the error-bounded MILP solver and in 88% of the cases found the same solution. The zero-sum game heuristic (ZS4), where the defender's utility is positively augmented by the attacker's cost, has even better scalability and provides a good trade-off between scalability and solution quality. The perfect information heuristic had the best scalability at the cost of finding worse strategies than ZS4. Nevertheless, it had a relative regret less than 3%. Therefore, this heuristic can be used to solve problems that are too complex for the algorithms that find strategies with lower regrets. To show the regret of heuristic approaches, we had to develop and solve the algorithms that compute optimal defense strategies.

All game-theoretic heuristics significantly outperformed the baseline approaches, such as placing honeypots uniformly or proportionally to the host type value. This shows that even if we are not able to find the exact solution to a complex game-theoretic model, finding good heuristics can be computed quickly and can result in acceptable outcomes for the defender.

For honeypot selection problem, we compared the game theoretic strategies with human strategies obtained from 45 responses of the survey participants. Their tasks were to decide where to allocate honeypots in one scenario and to choose an attack plan to perform in a different scenario. The average participants' defense strategy, despite being more exploitable than the optimal defense strategy against the worst-case attacker, performed better than the optimal defense strategy against the average participants' attack strategy. In other words, it is better to defend with human strategies against the human attackers. However, it neglects the aspect of player's long-term adaptation against the opponent's strategy. The attacker learn by trial and error which attack plans work and shift to it, which corresponds to the worst-case attackers. In such case the game theoretic strategies outperform the strategies of the participants.

We can use our algorithms to investigate general questions about security policy and illustrate the usefulness of the honeypots in securing the networks. Our analysis shows that there are interesting strategic questions in deciding how many and what type of honeypots should be used to harden a network. We showed that deploying fewer honeypots may result in a higher contribution to network security *and* in a higher honeypot deployment cost than intelligently deploying more honeypots. In the sensitivity analysis, we show that it is very important to accurately estimate the success probabilities of the attacker's actions, rather than the action costs or host type values. We also showed that the attacker's imperfect information can be exploited by the defender to increase his relative utility by up to 3% for medium-sized networks, and less for the larger networks.

Chapter 5

Threshold Selection Problem

In this chapter we study the problem of data exfiltration detection problem in the computer networks. We focus on the performance of optimal defense strategies with respect to an attacker's knowledge about typical network behavior and his ability to influence the standard traffic. Internal attackers know the typical upload behavior of the compromised host and may be able to discontinue standard uploads in favor of the exfiltration. External attackers do not immediately know the behavior of the compromised host, but they can learn it from observations.

This problem can be also modeled as a three-staged game with imperfect information presented in Chapter 3, where the network administrator selects the thresholds for the detector, while the attacker chooses how much data to exfiltrate in each time step. We present novel algorithms for approximating the optimal defense strategies in the form of Stackelberg equilibria. We analyze the scalability of the algorithms and efficiency of the produced strategies in a case study based on real-world uploads of almost six thousand users to Google Drive. We show that with the computed defense strategies, the attacker exfiltrates 2-3 times less data than with simple heuristics; randomized defense strategies are up to 30% more effective than deterministic ones, and substantially more effective defense strategies are possible if the defense is customized for groups of hosts with similar behavior.

5.1 Introduction

A common type of cyber attack is a *data breach* which involves the unauthorized transfer of information out of a system or network in a process called *information exfiltration*. Information exfiltration is a major source of economic harm from cyber attacks, including the loss of credit card numbers, personal information, trade secrets, unreleased media content, and other sensitive data. Many recent attacks on high-profile companies (e.g., Sony Pictures and Target) have involved large amounts of data theft over long periods of time without detection[1]. Improving strategies for detecting information exfiltration is therefore of great importance for improving cybersecurity.

[1] https://en.wikipedia.org/wiki/List of data breaches

We focus on methods for detecting information exfiltration activities based on detecting anomalous patterns of behavior in user upload traffic. An important advantage of this class of detection methods is that it does not require knowledge of user data or the ability to modify this data (e.g., to use honey tokens). To better understand the strategic aspects of anomaly detection and information exfiltration we introduce a two-player game model that captures the defender's and attacker's decisions in a sequential game. While we focus mainly on the information exfiltration example, note that raising alerts based on detecting anomalous behavior is a common strategy for detecting cyber attacks, so our model and results are relevant beyond just information exfiltration.

One of the novel aspects of our game model is that we consider both insider and outsider threats. A recent McAfee report [mca, 2015] states that 40% of serious data breaches were caused by insiders trusted by the organization, while the remaining 60% are due to outside attackers infiltrating the enterprise. Since both types of attacks are prevalent we consider both cases. There are significant differences between insiders and outsiders for information exfiltration. One difference is that an insider *knows* his typical behavior already and can use this knowledge to evade detection, while an outsider must *learn* this behavior from observation. A second difference is that insiders may be able to *replace* their normal activity with malicious activity, while an outsider's actions will be observed *in addition* to the normal activity. We model both of these key differences and examine how they affect both attacker and defender behavior in information exfiltration.

We introduce a sequential game model in which the objective of the attacker is to exfiltrate as much data as possible before detection, and the objective of the defender is to minimize the data loss before detection. The defender monitors the amount of data uploaded to an external location (e.g., Dropbox or Google Drive) and raises an alert if the traffic exceeds a (possibly randomized) threshold in a give time period. Some network companies use only uploaded data volume as feature to detect the data exfiltration. In our paper we follow this approach, however, our algorithm allows using more features as well. The defender is constrained to policies that limit the expected number of false positives that will be generated. The attacker chooses the amount of data to exfiltrate in each time period. We model both insider and outsider attackers, and both additive and replacing attacks. In the additive attack the total traffic observed is the sum of the normal user activity and the attack traffic, while in the replacing attack only the attack traffic is observed by the defender. Outsider attackers also receive an observation of the user traffic in each time period that can be used to learn the behavior pattern (and therefore infer something about the likely threshold for detection).

In his part, our first main contribution is the exfiltration game model that considers the differences between insider and outsider attackers. Our second contribution is a set of algorithms for approximating the optimal strategies for both defender and attacker players in these games. For outsider attackers, we use *Partially Observable Markov Decision Process* (POMDP) to model the learning process for the attacker. We also consider randomized policies for the defender, since it has been shown that static decision boundaries can be quickly learned ([Comesana et al., 2006]) and randomizing can mitigate successful attacks ([Lisý et al., 2014]). Our third main contribution is an experimental analysis of a case study based on real-world data from a large enterprise with 5864 users connecting to a

Google drive service for 12 weeks. We compute optimal strategies against different classes of attackers, and examine the characteristics of the strategies, the effects of randomization and attacker learning, and the robustness of strategies against different types of attackers. We show that with the computed defense strategies, the attacker exfiltrates 2-3 times less data than with simple heuristics; randomized defense strategies are up to 30% more effective than deterministic ones, and substantially more effective defense strategies are possible if the defense is customized for groups of hosts with similar behavior.

5.2 Related Work

Several previous works focus on detection and prevention of data exfiltration. A common approach is anomaly detection, e.g., a system can automatically learn the structure of standard database transactions on the level of SQL queries and raise alerts if a new transaction does not match this structure ([Lee et al., 2002, Fadolalkarim et al., 2016]). An alternative option is to create signatures of the sensitive data based on their content and detect if this content is sent out ([Liu et al., 2009]). The signatures should be resilient against the addition of noise or encryption, such as wavelet coefficients for multimedia files, which are resilient against added noise. Data exfiltration can also be partially mitigated by introducing automatically generating honey-tokes, a bait documents that rise alarm when are opened or otherwise manipulated ([Bowen et al., 2009]). These works do not consider volume characteristics of the traffic as means of detecting data exfiltration and do not study the learning process of the external attacker, which are the focus of this paper. A commonly studied option of exfiltration is to use a covert channel and hide the communication in packet timing differences of DNS requests ([Zander et al., 2007]). If the covert channel increases the volume of traffic to some service, the methods presented in this paper can help with its detection. More general data exfiltration motivations and best practices to protect the data are described in [Liu and Kuhn, 2010].

Data exfiltration and similar security problems have been previously studied in the framework of game theory. In [Liu et al., 2008] the authors propose a high level abstract model of insider threat in the form of partially observable stochastic game (POSG). They propose computing players' strategies using generic algorithms developed for this class of games, which have very limited scalability. The instance of the game they analyze in the case study focuses on data corruption and not exfiltration. Our work can also be seen as a special instance of POSG, but we provide more scalable algorithms to solve it and analyze the produced strategies in the context of data exfiltration.

Similar to our work, in [Laszka et al., 2016a] the authors investigates selecting thresholds for intrusion detection systems protecting distinct subsets of a network. The goal is to find optimal trade-offs between false positives and the likelihood of detection of an attack, which is simultaneously executed on a several subsets of the network. The attacker cannot decide what action to execute, only which systems to attack; nor he has an ability to learn the possible thresholds before the attack is conducted. In [Mc Carthy et al., 2016] the authors use POMDP to compute defender's optimal sequence of (imperfect) sensors to accumulate enough evidence whether data exfiltration over Domain Name System queries is happening

in the network or not. However, unlike our work, they assume non-adaptive attackers. The authors in [Lisý et al., 2014] investigated the effect of randomization of detection thresholds to strength of attacks and their overall cost to the defender. Our modeling of insider attacks is similar to this work. In contrast to our work, the attacker has perfect knowledge about the detector and the attacked system before the attack.

5.3 Game Theoretic Model

We model the problem of data exfiltration as a dynamic (sequential) game between the defender (network administrator) and the attacker trying to exfiltrate data over the network. We first discuss the basic setting of the game and focus on the fundamental differences between the insider and outsider and whether their activity is added to or replaces the normal traffic of the host. Then, we define the exact interaction between the attacker and the defender.

The **defender** monitors the volume of data uploaded by each network host[2] to a specific service over time, in time windows of constant length, e.g., 6 hours. His action is to select a detection threshold c from the set of available thresholds $\mathcal{A}_d$. If a host uploads higher data volume than c in a time window, an alarm is raised and the activity of that host is inspected by the administrator.

The **attacker** controls one of the users and tries to upload as much data as possible to the selected service before being detected. His actions are to choose the amount of data $a \in \mathcal{A}_a \subseteq \mathbb{N}_0$ he exfiltrates in the next time window. If this amount (possibly) combined with the host's standard activity is below c, the attacker immediately receives reward a and the defender suffers a penalty proportional to a. In this latter case, the attacker can act again in the following time window.

Since each **host** in a company may have different pattern of standard activity, the defender might want to set the threshold for each of them individually. However, this approach can be laborious in big companies and individual users rarely produce enough data for creating high quality models of their behavior. Therefore, it is common to create groups of hosts with similar behaviors and reason about these groups instead. In our models we refer to each group as a host type s from a set of all types $\mathcal{A}_c$. We assume both players know the probability $\mathcal{C}(s)$ that a randomly selected host in the network is of type s. Each host type is characterized by its common activity pattern in the form of the probability $P(o|s)$ that a host of type s transfers the amount of data $o \in O \subseteq \mathbb{N}_0$ in a time interval. We call these amounts observations, since they are the information observed by the external attackers.

The standard host's activity can sometimes surpass the selected threshold even without any attacker's activity and the host is still inspected. These *false positives* take a lot of time for the administrator to investigate and are typically a key concern in designing IDS. To capture this constraint we require the defender's strategies to have an expected number of false positives bounded by a constant FP.

[2]Hosts are non-strategic actors in the game considered to be part of the environment.

5.3.1 Outsider vs. Insider

The outsider is an external attacker who compromises a host in the computer network to exfiltrate data. Although the outsider may know what types (groups) there are in the company (secretaries, IT administrators, etc.), they often *do not know* which host type they compromised. However, they can observe the activity of the compromised host in each time window and update their belief about its type. Starting an aggressive exfiltration is likely not the best strategy, since once attacker surpasses a threshold, he is detected and the attack is stopped. However, conducing too much observation may cause that the host is disconnected or turned off before any exfiltration was conducted; or that the user's normal traffic surpasses the threshold, in which case the host is inspected and the attacker may be detected; or the data may become useless. We model this risk by discounting future rewards t time steps ahead with γ, where $\gamma \in (0, 1)$. The outsiders must cautiously weigh how much to exfiltrate at the current time step versus how long to learn the host type to increase future rewards. Typically, he would first emphasize learning with little data exfiltration, and proceed to more aggressive exfiltration when he is more certain about the host type.

The insiders are the regular users of the network and they *know* their host type and the deployed defenses. If the defender sets a fixed threshold for each host type, an insider can exfiltrate exactly at that threshold (we assume that the amount of data has to surpass the threshold to trigger the alarm). Such a defense strategy is not optimal, and the defender should minimize the insiders certainty about the threshold by randomization of her choices.

5.3.2 Additivity vs. Replacement

Consider a situation where the host uploads o MB and has set threshold c. Then the attacker can exfiltrate at most $c - o$, if he does not want to surpass the threshold. Additivity is important mainly for the external attacker operating on the host without its user's knowledge. However, we allow additivity even for the insider in our model so that we can analyze the effect of incomplete knowledge of the external attacker with all other conditions equal. Assuming that the attacker completely replaces the existing host traffic with the exfiltration is more natural for the insider. However, even the external attacker can, in principle, throttle or to completely block the standard user's traffic to increase his own bandwidth for exfiltration. We analyze combinations of scenarios when the attacker is insider/outsider and when the user's normal traffic is and is not present.

5.3.3 Formal Definition of Game Model

We have introduced the following components: $\mathcal{A}_c$ is the set of host types and $\mathcal{C}(s)$ the probability of occurrence type $s \in \mathcal{A}_c$; O (resp. $\mathcal{A}_a$) is the set of possible amounts of data that the hosts (resp. attackers) can upload; $P(o|s)$ describes host's standard activity; $\mathcal{A}_d$ is the set of thresholds the defender can choose; FP is the defender's maximal false positive rate; γ is the discount factor.

In our model, we assume that the network administrator models the user's normal traffic using discrete representation, e.g., histograms. In that case, the attacker and defender's

action are also discrete, as they have no incentive to choose actions between the discrete values.

Defender's Strategy

We allow *mixed* (or randomized) strategies in form of $\delta_d(sc)$, where the defender chooses a probability distribution of thresholds c given host-type s. A valid defender strategy δ_d must satisfy the false positive constraint $\sum_{s \in \mathcal{A}_c} \sum_{c \in \mathcal{A}_d} \delta_d(sc)\mathcal{C}(s)FP(c|s) \leq FP$, where $FP(c|s) = \sum_{o \in O: o > a} P(o|s)$ is type s's amount of false positives if threshold is at c.

Attacker's Strategy

In the course of the attack, the attacker follows a policy which prescribes what action he should take when he played actions $a_1, \ldots, a_k$ and saw observations $o_1, \ldots, o_k$ so far ([Braziunas, 2003]). We assume, that the defender chooses his threshold strategy first, and the attacker acts afterwards, knowing the defender's strategy. In such a case, the attacker acts only against the nature, without adversarial actor, and Partial Observable Markov Decision Processes (POMDPs) can be used to reason about (approximately) optimal attacker's policies for the attacker. In the POMDP the attacker is not required to remember the entire history of his actions and observations. Instead, he can capture all relevant information he has acquired in the course of the interaction in the form of a belief state $b \in \Delta(\mathcal{A}_c \times \mathcal{A}_d)$, which is a probability distribution over possible host types and threshold settings. We can then define attacker's policy based on his belief as $\pi_a : \Delta(\mathcal{A}_c \times \mathcal{A}_d) \to \mathcal{A}_a$. In the course of the interaction, the attacker keeps track of his belief b using a Bayesian update rule when he takes the last action and observation into account. Based on his current belief, he chooses action $\pi_a(b)$ to play. We denote the set of all belief-based policies as Π_a.

Note that the insider knows the host type from which he exfiltrates the data (it is his own host machine), therefore, there is no need to update belief based on the observation. Therefore, the insider chooses an action from $\mathcal{A}_a$ that he plays in all time windows. Mixed strategies are probability distribution among these choices.

Utilities

We define the attacker's expected utility as $u_a(\delta_d, \pi_a)$, which is the discounted total expected amount of exfiltrated data using policy π_a against the defense strategy δ_d. We define the defender's utility as $u_d(\delta_d, \pi_a) = -Cu_a(\delta_d, \pi_a)$, where $C > 0$. That means, that players have opposing objectives and their payoffs are proportional. Typically $C > 1$, which means that the defender suffers more than the attacker gains. For simplicity, we ignore the attacker's and defender's individual action costs.

Because of the linearity axiom ([Straffin, 1993]), multiplying the defender's payoffs with with a constant $\frac{1}{C}$ will not change the player's strategies, and the game becomes a zero-sum game. In zero-sum games, all the attacker's best responses have the same expected utility to the defender and the attacker, therefore, there is no need to distinguish between specific best responses as in SSE. Because we use approximative algorithm to compute the attacker's

policy, we focus on finding ϵ-optimal SE. The defender's strategy in ϵ-SE guarantees, that the defender's utility cannot be improved by a factor of more than $1 + \epsilon$ in the exact SE.

Definition 21 (ϵ-optimal Stackelberg equilibrium (ϵ-SE))**.** *Let* $\epsilon \in (0, 1]$. *Solution profile* (σ^*, π^*) *where* $\pi^* \in BR_a(\sigma^*)$ *belongs to* ϵ*-SE, if* $\forall \sigma \in \Sigma, \forall \pi \in BR_a(\sigma) : \frac{u_d(\sigma,\pi) - u_d(\sigma^*,\pi^*)}{|u_d(\sigma^*,\pi^*)|} \leq \epsilon$.

Note, that we use *multiplicative* definition of approximate solution concept ([Feder et al., 2007]), rather then more typical *additive* approximation. In our opinion, the multiplicative approximation is slightly more reasonable for our domain. However, the algorithm can be easily modified to return additive ϵ-SE.

5.4 Algorithms

In this section, we present two algorithms. First algorithm computes the exact SE against the insider. Second algorithm computes ϵ-SE against the outsider.

5.4.1 Optimal Defense Strategy Against Insiders

Since we assume that the insider knows from which user type he exfiltrates data (they have complete information), we can model the interaction between the attacker and each host type as a normal-form game, where the attacker chooses a probability distribution over actions $\mathcal{A}_a$ for each host type and the defender chooses probability distribution over thresholds $\mathcal{A}_d$ for each host type. Because false-positive rate is a global constraint across all host types, we must compute the optimal defense strategy for all host types globally and not for each host type independently. We formalize the game between all host types and the attacker as one problem by extending the zero-sum normal-form linear program (LP) [Shoham and Leyton-Brown, 2008] with multiple host types and a false-positive constraint.

$$\min_{r_d} U_a \tag{5.1a}$$

$$\text{s.t.} \quad : (\forall s \in \mathcal{A}_c, \forall a \in \mathcal{A}_a) : \sum_{c \in \mathcal{A}_d} u_a(sca) r_d(sc) \leq U_{a,s} \tag{5.1b}$$

$$\sum_{s \in \mathcal{A}_c} \mathcal{C}(s) U_{a,s} \leq U_a \tag{5.1c}$$

$$(\forall s \in \mathcal{A}_c) : \sum_{s \in \mathcal{A}_c} r_d(sc) = 1 \tag{5.1d}$$

$$(\forall s \in \mathcal{A}_c \forall c \in \mathcal{A}_d) : r_d(sc) \geq 0 \tag{5.1e}$$

$$\sum_{s \in \mathcal{A}_c} \sum_{c \in \mathcal{A}_d} \mathcal{C}(s) r_d(sc) FP(c|s) \leq FP \tag{5.1f}$$

The variables in the LP are: $r_d(sc), U_a$ and $U_{a,s}$. $r_d(sc)$ denotes the probability of sequence sc, where s is the host type and c is the threshold. The objective (5.1a) minimizes the

attacker's expected utility U_a, which consists of expected utilities $U_{a,s}$ of each type, weighed by its probability (5.1c). Constraints (5.1b) ensure that the attacker plays a best response in each host type against the given defense strategy; (5.1d) and (5.1e) ensure that the defender's strategy is a proper probability distribution; and (5.1f) ensures that the strategy meets the required false-positive rate. The solution of the LP will return the defender's realization plan in variables r_d. This LP does not return the attacker's strategy, however, since we seek for the attacker's pure strategy in SE, finding an action that yields maximal utility to the defender is just an evaluation of a polynomial formula.

The attacker's expected payoff in terminal node with history sca we compute as follows:

$$u_a(sca) = \begin{cases} \frac{a}{1-\gamma} \text{ if } c \geq a \\ 0 \text{ otherwise.} \end{cases} \tag{5.2}$$

The value $\frac{a}{1-\gamma} = a + \gamma a + \gamma^2 a + \ldots$ is the attacker's cumulative discounted reward for selecting $a \leq c$. For the attacker with additivity the expected payoff is computed as follows:

$$u_a(sca) = \frac{aP[o + a \leq c|s]}{1 - \gamma P[o + a \leq c|s]} \tag{5.3}$$

where $P(o + a \leq c|s) = \sum_{o \in O: a+o \leq c} P[o|s]$ is the probability that the user's action o combined with the attacker's action a is below threshold c for type s. The equation, similarly as the previous one, sums the attacker's expected utility for action a with γ discount and takes into account, that with probability $P[o+a \leq c|s]$ the attacker surpasses threshold c. To compute the defender's pure strategy, we replace Equation 5.1e with $(\forall s \in \mathcal{A}_c \forall c \in \mathcal{A}_d) : r_d(sc) \in \{0, 1\}$, which forces the defender's realization plan for sc to be either zero or one.

5.4.2 Optimal Defense Strategy Against External Attacker

The outsider observes the activity of the host in an attempt to learn and infer its type. Due to the learning process, the strategies of the attacker are more complex, compared to the insider case, as the strength of the attack can vary over time. We can reason about attacker's behavior under this uncertainty using Partially Observable Markov Decision Processes (POMDPs) and his optimal, best-response strategy can be computed by algorithms for solving POMDPs. Originally, POMDPs were designed to reason about actions of a single decision maker. However, since the defender only decides the initial belief of the POMDP and the defender then has no influence on the dynamics of the system, we can extend the POMDP framework to solve our game-theoretic problem.

The POMDP framework assumes that in every time step, the player chooses an action and receives an observation from the environment as a result. Based on this observation he updates his belief over the possible current states of the environment. Additionally, in each time step the player obtains a reward which depend on the state of the environment and the action chosen. A solution of the POMDP is a policy which prescribe an action to use given every possible belief state. Here, we extend a well-established algorithm for solving infinite-horizon discounted POMDPs, Heuristic Search Value Iteration (HSVI) (from

[Smith and Simmons, 2004]) to find an ϵ-Stackelberg Equilibrium, with key ideas inspired by [Horák and Bošanský, 2016]. The main idea of the algorithm is to iteratively compute the attacker's and defender's best response strategies, which will eventually converge to a Stackelberg equilibrium. In this chapter, we do not distinguish different versions of the SE (strong, weak, etc.), because in zero-sum games all versions are equivalent. In this case,

In this section, first, we define POMDP models formally; then we explain the main ideas of the HSVI algorithm; and lastly, we present our contribution, the Adversarial HSVI algorithm, aimed to find ϵ-SE in our game.

POMDP Model

Given the defender's strategy δ_d represented as realization plan r_d, we define an infinite-horizon discounted POMDP model as a tuple $\langle S, A, O, T, R, P, \gamma \rangle$, where:

- $S = \mathcal{A}_c \times \mathcal{A}_d$ is set of states, where each state $\alpha \in S$ is defined as $\alpha = (c_\alpha, s_\alpha)$, where s_α is host-type and c_α is the chosen detection threshold. We also define a terminal state α_T, which denotes that the attacker got detected and the attack was deflected.
- $A = \mathcal{A}_a$ is the set of attacker's actions;
- O is the set of observations about the traffic on host attacker tries to exfiltrate;
- $T[\alpha'|a, \alpha]$ is the probability that action a in state α leads to new state α'. In our case, when additivity is considered $\forall \alpha \in S \setminus \{\alpha_T\} : T[\alpha'|a, \alpha] = P(a + o \leq c_\alpha|s_\alpha)$, and $T[\alpha'|a, \alpha] = 1 - P[a + o \leq c_\alpha|s_\alpha]$. If there is no additivity, then $\forall \alpha \in S \setminus \alpha_T : T[\alpha'|a, \alpha] = 1_{a \leq c_\alpha}$ and $T[\alpha'|a, \alpha] = 1_{a > c_\alpha}$ otherwise, where $1_A = 1$ if A is true and $1_A = 0$ otherwise is the indicator function.
- $R(\alpha, a, \alpha')$ is the immediate reward the attacker obtains for performing action a in state α leading to α'. In our case $R(\alpha, a, \alpha') = a$ had the attacker not been detected yet, $R(\alpha, a, \alpha') = 0$ otherwise;
- $P[o|a, \alpha]$ is the probability of observing $o \in O$ when action a is taken in state α. In our case $P[o|a, \alpha = (s_\alpha, c_\alpha)] = P[o|s_\alpha]$.
- $\gamma \in (0, 1)$ is the discount factor.

With $\mathcal{B}$ we denote the attacker's **belief space**, i.e. the set of all probability distributions over the states S. We derive the initial belief $b_0 \in \mathcal{B}$ according to the prior distribution over the host types $\mathcal{C}(s)$ and the strategy of the defender, i.e. $b_0(s) = r_d(sc)\mathcal{C}(s)$ for state $\alpha = (s, c)$.

Adversarial HSVI (A-HSVI)

Recall that the initial belief of the POMDP problem, $b_0(\alpha) = r_d(sc)\mathcal{C}(s)$, can be directly mapped to the defender's realization plan r_d (and vice versa). Therefore, we search such initial belief b_0 for the defender, that meets the maximal false positive constraint and minimizes the attacker's expected utility (POMDP upper bound value at b_0). In high level, our approach iteratively alternates between selecting a promising initial belief b_0 (strategy for the defender) and solving POMDP at that belief b_0.

In Algorithm 9 is the pseudocode of Adversarial HSVI algorithm (A-HSVI). The outline of the algorithm is very similar to the original HSVI algorithm, as presented in Section 2.2.1.

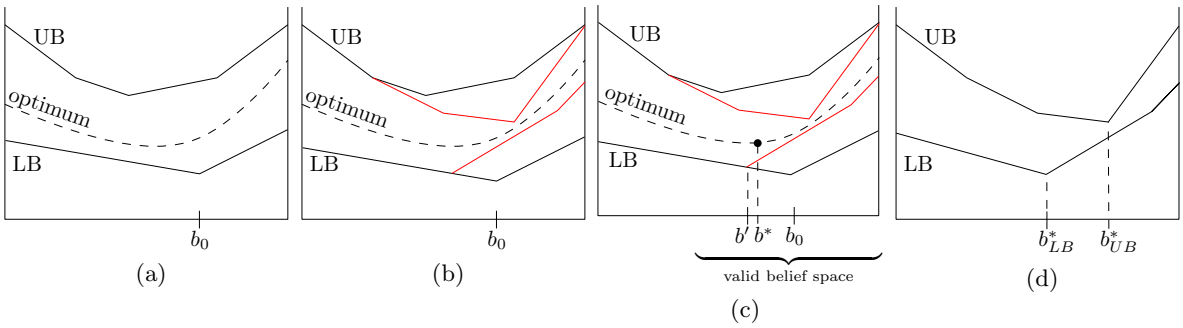


Figure 5.1: Original and Adversarial HSVI algorithm: (a) initial upper bound (UB) and lower bound (LB) on the (unknown) optimal value function v^*. HSVI aims to minimizes the gap between UB and LB in the initial belief b_0. (b) After one HSVI iteration, tighter approximation using LB and UB is computed. (c) In Adversarial HSVI the defender chooses a new belief b' where LB has minimal value in every iteration. (d) A possible scenario when algorithm is converged and a conservative strategy for the defender, based on b^*_{UB}, is returned.

In detail, to find initial belief in ϵ-SE and the strategy r_d for the defender, the Adversarial HSVI algorithm extends the original HSVI algorithm in two ways (the A-HSVI algorithm is illustrated in Figure 5.1). First, instead of having fixed initial belief b_0, our algorithm chooses a new belief b' in every iteration. This belief, $b' = \text{argmin}_b\, LB(b)$, is chosen to minimize attacker's lower bound value. Second, we limit the depth D of the HSVI simulation by $\sqrt{iter}$, where $iter$ is the current iteration number. We do this to emphasize the exploration of the belief space first, and then focus on the computation of more accurate bounds later on (Figure 5.1). The rest of the algorithm follows the ideas of the original HSVI algorithm.

Figure 5.1 illustrates one iteration of the algorithm. In Figure 5.1a are presented known upper and lower bound, the unknown optimal value and current initial b_0 corresponding to the current defender's strategy. After one HSVI simulation into depth $\sqrt{iter}$, the upper and lower bounds are updated, as illustrated with red lines in Figure 5.1b. Next, the defender chooses new initial belief b' that minimizes the attacker's value, therefore, the lowest value of the lower bound that does not violate the false-positive rate (denoted as valid belief space), as depicted in Figure 5.1c. This process is repeated until the termination condition of the algorithm is met. Let the algorithm terminate with upper and lower bounds as depicted in Figure 5.1d and $b_{LB} = \text{argmin}_b\, LB(b)$ and $b_{UB} = \text{argmin}_b\, UB(b)$ be the beliefs with minimal value of lower and upper bounds. We ensure that the algorithm finds ϵ-SE, by terminating when the defender's and attacker's strategies have maximum relative error ϵ and then we return a secure strategy implied by belief b_{UB}. The attacker can guarantee that he will obtain at least $LB(b_{LB})$, while the defender can guarantee that he will not lose more than $UB(b_{UB})$. Based on these numbers, we compute an upper bound on the relative improvement of defender's strategy (i.e. if he plays b_{LB} instead of b_{UB}) as $\frac{UB(b_{UB}) - LB(b_{LB})}{UB(b_{UB})}$.

Algorithm 9 Adversarial HSVI (A-HSVI).

procedure FIND ϵ-SE(ϵ)
  $b_{LB} \leftarrow$ minimal Lower bound
  $b_{UB} \leftarrow$ minimal Upper bound
  $iter \leftarrow 1$
  while $\frac{UB(b_{UB})-LB(b_{LB})}{UB(b_{UB})} > \epsilon$ **do**
    Explore($b_{LB}, 1, iter$)
    $b_{LB} \leftarrow$ minimal Lower bound
    $b_{UB} \leftarrow$ minimal Upper bound
    $iter \leftarrow iter + 1$
  end while
  return $r_a(sc)$ induced by b_{UB}
end procedure

Algorithm 10 Subroutine of A-HSVI algorithm.

procedure EXPLORE($b, depth, iter$)
  if $depth < \sqrt{iter}$ **then**
    $a^* \leftarrow$ best action to explore
    $o^* \leftarrow$ best observation to explore
    $b' \leftarrow \tau(b, a^*, o^*)$
    Explore($b', depth + 1, iter$)
  end if
  UpdateLB()
  UpdateUB()
end procedure

Proposition 1. *Adversarial HSVI (Algorithm 9) returns ϵ-SE.*

Proof. Without loss of generality, we assume the game is exactly zero-sum (i.e., $C = 1$). When the algorithm terminates and returns $\sigma(sc)$ induced by b_{UB}, we know that the best response of the attacker to the defender's strategy induced by b_{UB} cannot gain more than $UB(b_{UB})$, hence the defender's cost $-u_d(\sigma(sc), BR_a(\sigma(sc)) \leq UB(b_{UB})$. If the defender played any alternative strategy σ', we know that the attacker would always be able to exfiltrate at least $LB(b_{LB})$ by definition of b_{LB}, hence $-u_d(\sigma', BR_a(\sigma')) \geq LB(b_{LB})$. If the termination condition is satisfied, $\frac{UB(b_{UB})-LB(b_{LB})}{UB(b_{UB})} \leq \epsilon$. Therefore, it is sufficient to show that the relative error of the computed strategy

$$\frac{u_d(\sigma', BR_a(\sigma')) - u_d(\sigma(sc), BR_a(\sigma(sc))}{|u_d(\sigma(sc), BR_a(\sigma(sc))|} \leq \frac{UB(b_{UB}) - LB(b_{LB})}{UB(b_{UB})}.$$

Since the defender's utility is always negative, we know that

$$|u_d(\sigma(sc), BR_a(\sigma(sc))| = -u_d(\sigma(sc), BR_a(\sigma(sc)).$$

Hence, the above is equivalent to

$$1 - \frac{u_d(\sigma', BR_a(\sigma'))}{u_d(\sigma(sc), BR_a(\sigma(sc))} \leq 1 - \frac{LB(b_{LB})}{UB(b_{UB})} \quad \text{and} \quad \frac{-u_d(\sigma', BR_a(\sigma'))}{-u_d(\sigma(sc), BR_a(\sigma(sc))} \geq \frac{LB(b_{LB})}{UB(b_{UB})}.$$

This is true, because from left to right in the last inequality, the nominator can only decrease and the denominator can only increase. □

5.5 Real-world Data

From a large network security company we obtained anonymized data capturing the volumes of upload of 5864 active Google drive users uploaded during 12 weeks. For each user we

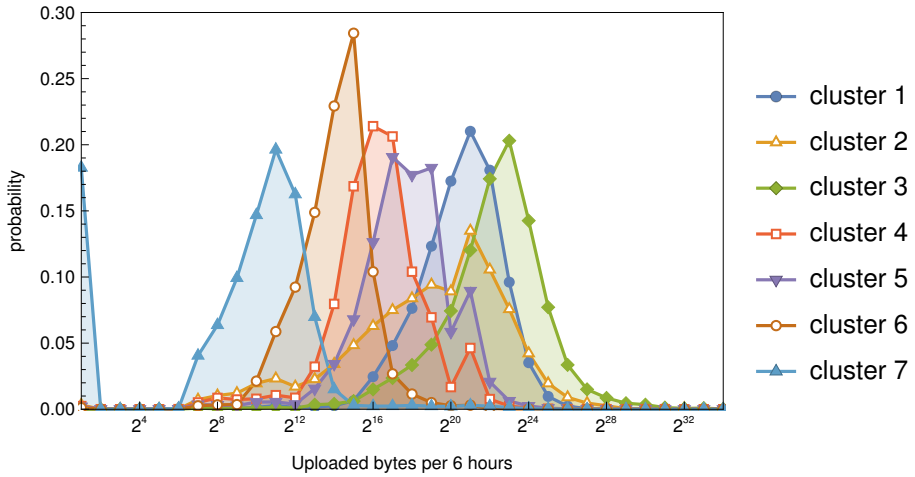


Figure 5.2: Mean upload size histograms of the identified clusters of users.

computed the amount of data that the user uploads in 6 hour windows. Next, we created histograms showing how often the user uploaded certain number of bytes per 6 hours, which can be understood as user's upload probability distribution.

We used the Partitioning Around Medoids algorithm to find clusters of similar behavior of the users where similarity was measured by Earth Mover's Distance [Rubner et al., 2000] metric. In Figure 5.2 we present 7 histograms corresponding to user's average behavior in each cluster and their relative membership. The clusters (in order) contain 25.6%, 5.5%, 17.2%, 8.9%, 11.7%, 11.5% and 19.6% of the total users.

5.6 Experiments

We now demonstrate our framework for a case study based on the real-world data. In all settings we choose: false positive rate $FP = 0.01$, the relative error of the strategy $\epsilon = 0.2$, and discount factor $\gamma = 0.9$. Decreasing the ϵ error or FP, or increasing discount γ resulted in too complex problems for A-HSVI to solve in two-hour time limit, as described in Section 5.6.5. We chose the set of attacker actions $|\mathcal{A}_a|$, the defender's thresholds $|\mathcal{A}_d|$, and the observations $|O|$ to be the set of $\{2^0, 2^2, 2^4, \ldots, 2^{34}\}$ bytes.

The structure of this section is as follows: In Section 5.6.1 we evaluate how much an optimal attacker can exfiltrate under various condition, in Section 5.6.2 we present a visualization of what optimal defense strategies look like, in Section 5.6.3 we evaluate defender strategies against different attacker models. In Section 5.6.4 we show how the presence of additivity influences the defense strategy, and finally, in Section 5.6.5 we present scalability results for computing the defense strategies.

Table 5.1: Attacker's utility for different scenarios: (columns) insider or outsider, with replacement or additivity and (rows) whether the defender plays optimal pure, optimal mixed or baseline strategy.

	Insider [MB]		Outsider [MB]	
	Replacement	Additivity	Replacement	Additivity
Mixed defense	23.71	3.56	(18.68, 24.81)	(3.11, 3.43)
Pure defense	33.58	5.03	(24.17, 29.87)	(3.78, 4.49)
Baseline single-threshold (mixed)	68.86	14.54	(65.45, 68.86)	(13.96, 14.56)
Baseline single-quantile (mixed)	65.32	12.31	(54.85, 57.58)	(10.39, 10.84)
One cluster (mixed)	63.29	12.95	N/A	N/A

5.6.1 Defender and Attacker Utilities

We now examine how much different attacker models can exfiltrate in our case study. In Table 5.1 we present a summary of the attacker's expected utilities (the attacker maximizes and the defender minimizes the value) for different types of the attackers. The columns indicate whether the attacker is an insider or on outsider and whether the attack is with replacement or with additivity. The rows indicate whether the defender plays a mixed or pure defense strategy, or a baseline defense. We present utilities against the outsider as the minimal lower bound and the minimal upper bound values from A-HSVI algorithm.

Note that the insider with replacement can exfiltrate up to about 6 times more compared to the insider with additivity. In the case with additivity, the typical traffic of a user is added to the traffic of the attacker; hence, the attacker must choose a less aggressive strategy (i.e., upload less data) so that the total data upload does not exceed the threshold. Although the attacks with additivity are disadvantageous to the attacker, in some cases the additivity is unavoidable, e.g., when different detectors detect whether the user runs standard processes (which generate a standard traffic). Next, we see that the user type uncertainty caused up to 14%–27% decrease in the utility (computed from the lower bounds). To verify that this outcome does not rely on the fact that some of the host-types have higher prior probability than the others, we additionally ran experiments with a uniform prior probability of the users, and the outsider had about 16% lower utility compared to the insider. Finally, note that if the defender must choose a pure strategy (e.g., due to practical deployment reasons) the amount of data exfiltrated by the attacker can be 24%-42% higher compared to randomized strategies.

We compare our strategies against two baseline approaches: (i) *single-threshold* and (ii) *single-quantile*. The single-threshold sets the same threshold value for all host types such that the total false-positive rate requirement is satisfied. Since we have discrete thresholds, the defender's strategy randomizes between two consecutive thresholds to reach the exact $(1 - FP)$ false positive rate. The experimental results show that the attacker can exploit this straightforward strategy and can exfiltrate about 3-times more data than against the optimal solution. The main reason is that this strategy chooses thresholds for passive users (e.g., cluster 7) too high, and the attacker easily exfiltrates from them. In the single-quantile strategy, the defender sets for each host type a threshold at quantile $(1 - FP)$ of

their upload probability distribution (again, randomizes between two consecutive discrete values to reach exact quantile $(1 - FP)$). This strategy yields a slightly better utility than the previous one; however, it sets the thresholds too high for the users with a large data upload (e.g., cluster 3) to satisfy the false-positive constraint.

Additionally, to show that it is worth developing different defense strategies for different user types, we computed the optimal defense strategy where all users belong to one cluster (instead clustering them into 7 clusters). The defender's expected utilities were 2–3 times worse than the optimal defense strategy with clustered users. Since there is only one cluster, the attacker does not need to learn the cluster of the attacked host. Therefore, there is no difference between insiders and outsiders in this case.

5.6.2 Defender's and Attacker's Strategies

Insiders

In this section we present the computed optimal strategies of the defender against the insiders without additivity. Figure 5.3a shows the defender's (cumulative) probability of selecting thresholds (x-axis) for each host type against the insider with replacement. The cumulative distributions show the probability that attacker exfiltrating data at a certain rate is detected. In Figure 5.3b is the attacker's expected utilities for different attacks on different host types given that the defender uses the strategy depicted in Figure 5.3a. The defender's strategy is computed in such a way that it makes the attacker indifferent between intervals of actions (e.g., for host type 1 the attacker is indifferent between actions 2^{22} through 2^{30}), which is typical for game theoretic strategies. The attacker's best response is to choose any of the attack actions that have the highest expected utility. Note that the host types with the highest activity (e.g., host types 3 and 1) result in the highest expected reward for the attacker. Therefore, these hosts should be monitored thoroughly to avoid potentially large data loss.

Figure 5.3c shows the optimal strategy of the defender when restricted to pure strategies. For host types 3, 6 and 7, corresponding to the largest and the two smallest mean upload sizes, the defender chooses threshold 2^{22}. The threshold of 2^{24} is chosen for all the other types. The expected utility of the attacker depicted in Figure 5.3b has peaks up to 5 MB, since with the pure defense strategies it is impossible to make the attacker indifferent between multiple actions. By randomizing non-trivially between multiple thresholds the defender can reduce his data loss.

The defense strategy against the insiders with additivity (in Figure 5.3e) is different to the previous ones. The optimal defense strategy lowers the thresholds of the most active users (host types 1 and 3) to restrict their large loss, and increases the thresholds of the less active users to compensate the false positives. Figure 5.3f shows the attacker's utilities for each pure strategy, however, we clusters 1 and 3 are still most exploitable.

Outsiders

The optimal defense strategies against the outsider with additivity (in Figure 5.4a) (resp. with replacement in Figure 5.4b) are more complex than the strategies against the insiders.

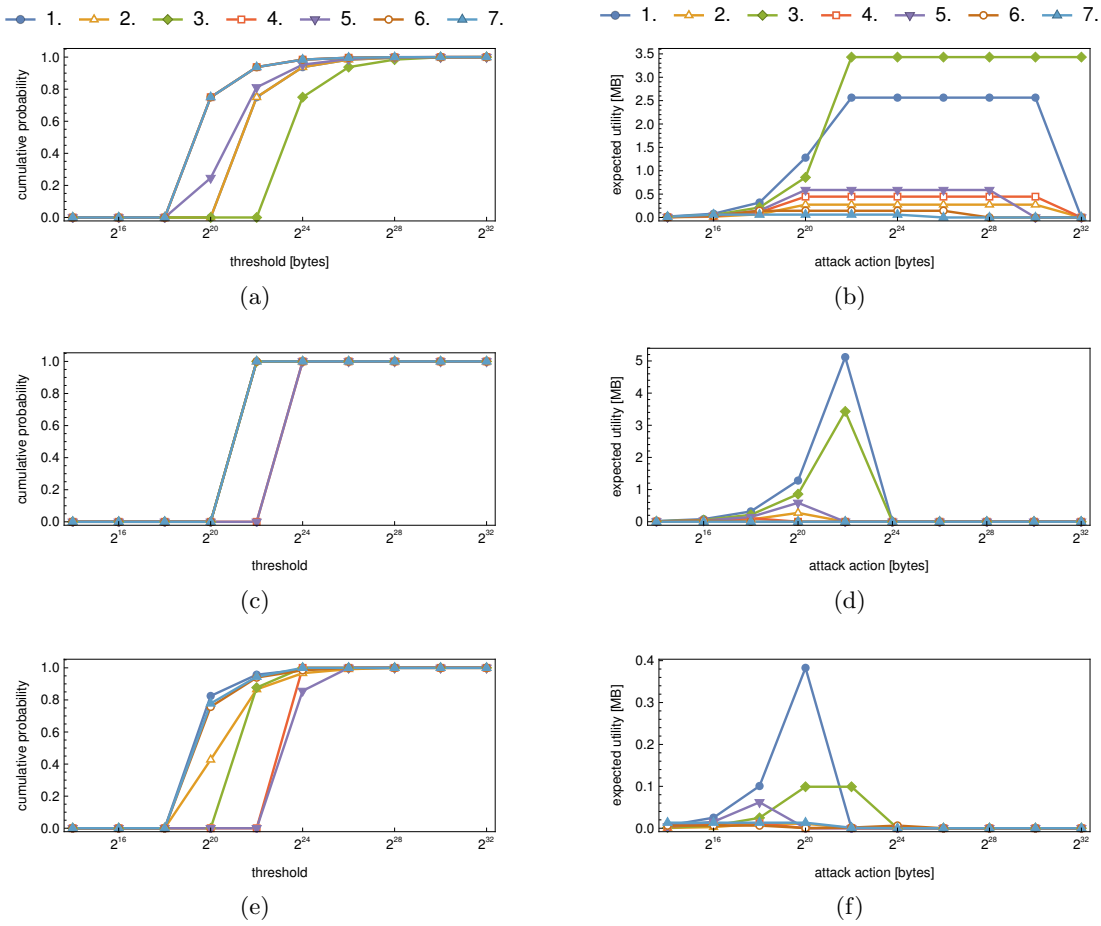


Figure 5.3: The defender's strategies and the attacker's expected utilities for individual attack actions for: (a,b) mixed defense strategy against insider with replacement; (c,d) pure defense strategy against insider with replacement; and (e,f) mixed defense strategy against insider with additivity.

The strategies consider how the attacker attacks and learns from the observations each time step as well as the fact that the value of the data decreases over time due to the discount factor. None of the above had to be considered against the insider attacker.

We attempt to explain how the defense strategy takes into account these aspects by examining the attacker's long-term behavior. In Figure 5.4c we show the average action in every time step *given* that the observations are drawn from a given host type (different curves) and that the attacker was not detected until the previous time step of the attacker with additivity. First, the attacker plays safe action 2^{20} since he has now other information about the host types except his prior belief about them. After receiving the first observation the attacker updates the probability distribution over the host types and strengthens his attack accordingly. On the other hand, the defender knows that the first attack action

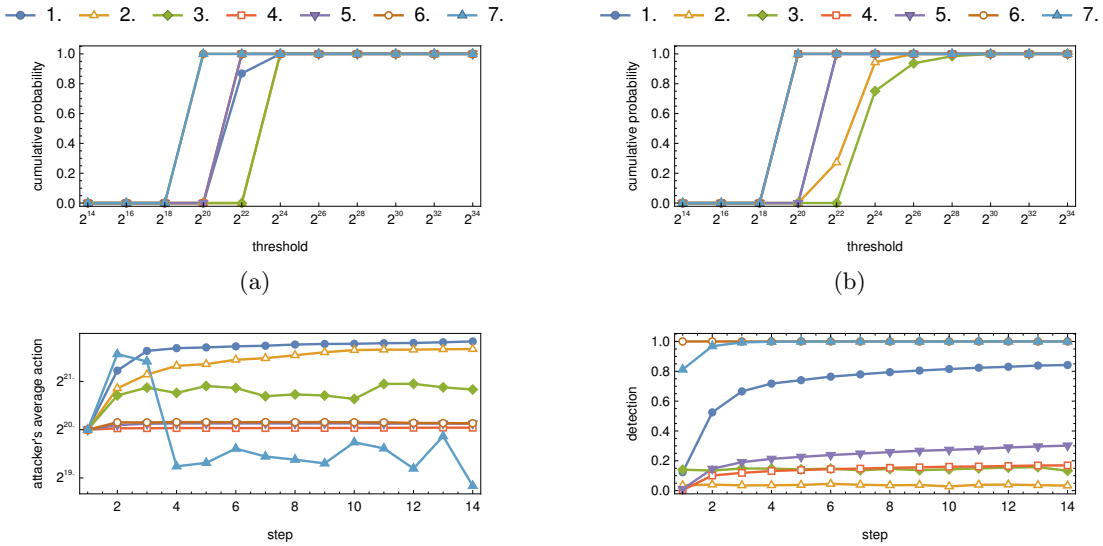


Figure 5.4: Defender's strategy against outsider with (a) additivity and (b) replacement; the (near) optimal attacker's response to (a) characterized by (c) the average action played at certain time step and (d) the probability of reaching the time step when attacking the given host type.

is going to be 2^{20}, therefore, he prefers setting the thresholds for host types 6 and 7 to 2^{20} (which is lower than it was against the insider attackers), which detects the attacker in 31.3% of the hosts during the first attack action (in Figure 5.4d are shown the probabilities that the attacker is detected until a given time step). Not only does it increase the detection probability, but setting the same threshold for both types also slightly reduced the information that the attacker obtains. Since thresholds for host types 6 and 7 are lowered, to compensate the false positives the defender increases the thresholds for host types 2 and 3. The optimal attacker is able to exfiltrate from host type 2, 4 and and 5 that are almost never detected but represent very low percentage of the users. The greatest risk present host type 3, who is detected with lower probability than 0.2 and 17.2% of the users belongs to it, and a host type 1 covering 25.6% of the users. However, in the latter case the probability of detecting the attacker is almost 0.8 at 9th time window.

This example shows how sophisticated the defender's strategy against the outsider attacker can be as they must consider a complex behavior of the attacker and all possible sequences of observations and attacks. To minimize the loss, the defender aims to detect the attacker as soon as possible. Using the presented defense strategies, the attacker that attacks host types 1, 6 and 7 (which corresponds to 56.7% of the users) is detected with higher probability than 0.5 by the third time window.

Table 5.2: Discounted expected amount of data that the attacker can exfiltrate if defense strategy (row) is optimized against the attacker in the "Strategy against" column, played against the different type of attackers (columns). The intervals for the outsiders represent lower and upper bounds of the optimal value.

	Strategy against	insider, additive	insider, repl.	outsider, additive	outsider, repl.
mixed	insider, add	3.56 MB	26.58 MB	(3.56, 4.84) MB	(23.51, 29.5) MB
	insider, rep	5.91 MB	23.71 MB	(5.59, 8.33) MB	(23.71, 32.72) MB
	outsider, add	3.69 MB	27.59 MB	(2.67, 3.32) MB	(20.24, 27.59) MB
	outsider, rep	3.71 MB	27.59 MB	(2.42, 3.4) MB	(18.59, 26.35) MB
pure	insider, add	5.03 MB	33.59 MB	(3.58, 4.43) MB	(24.54, 29.95) MB
	insider, rep	5.03 MB	33.59 MB	(3.58, 4.43) MB	(24.54, 29.96) MB
	outsider, add	5.03 MB	33.59 MB	(3.72, 4.49) MB	(24.55,29.95) MB
	outsider, rep	5.03 MB	33.59 MB	(3.58, 4.42) MB	(24.17, 29.87) MB

5.6.3 Different Attacker Models

Computing a defense strategy against the insiders can be done using linear programming, which is computationally more efficient than the Adversarial HSVI algorithm used against the outsiders. It raises a question whether the efficiently computed strategies against the insiders could be used also against the outsider and at what utility loss for the defender. In Table 5.2 we show the expected attacker's utilities if different defense strategies are applied against the different attacker models. The attacker of different types (table columns except for the column "Strategy against") plays the best response according to their model against different defender's strategies (rows). Each defense strategy was optimized against the attacker listed in column "Strategy against". We can use these values to compute the defender's *regrets* for optimizing the defense strategy against a different attack model than according to which the real attacker attacks. For example, if we optimize the strategy against the insider with additivity (first row), while the attacker is an outsider with additivity (fourth column), than the defender's regret is at most 36% (computed as $\frac{4.84-3.56}{3.56}$), based on the upper bound of the Adversarial HSVI.

Now, let us analyze the defender's regret for deploying strategies and optimizing against the insider attackers, while in reality the outsiders attack. If the defender correctly guesses or estimates the attacker's additivity/replacement, then in the worst case the defender can experience regret on average of 34% (with additivity) and 25% (with replacement). However, computing the worst-case regret (due to the A-HSVI bound gap), the defender can improve by up to 67% (with additivity) and 76% (with replacement). Computing the more precise values requires much longer simulation of the A-HSVI algorithm. In our personal opinions, the average regrets around 30% are not too high and in the case that the A-HSVI algorithm does not scale for the required size of the problem, it can be used.

In the case the defender knows the probability distribution of the attack models, this cross-strategy table analysis can be used to compute choose the strategy that yields lowest overall regret to the defender.

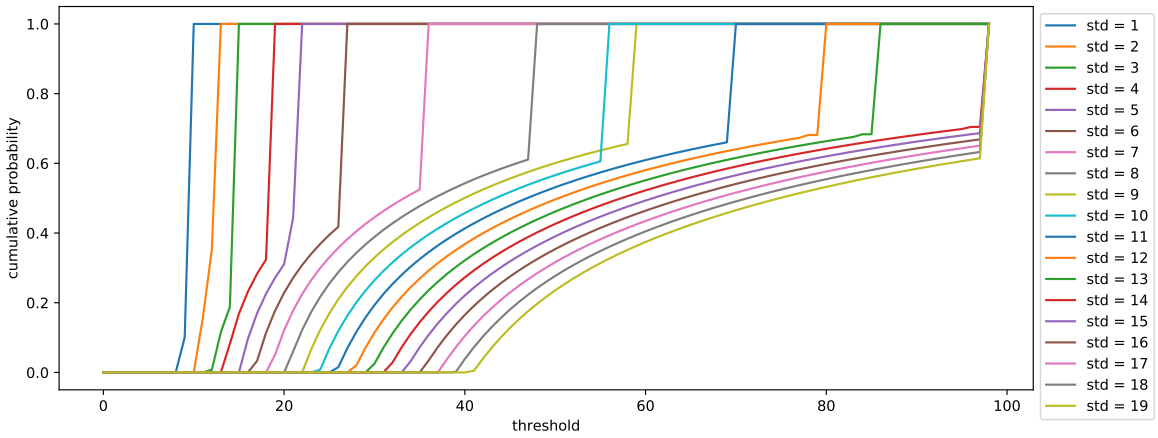


Figure 5.5: Optimal defense strategies for users with standard activity of Normal distribution with $\mu = 2^{20}$ and standard deviations $\sigma = 2^i$, where i is given parameter.

5.6.4 Effect of the Additivity

In this experiment, we analyze how the uncertainty of the user's behavior affects the defender's optimal strategy. In this analysis, we model only one user whose behavior follows the normal distribution with the mean equal to 10 and the standard deviation varying from 1 to 19. For the purpose of this experiment, we let $|\mathcal{A}_a| = |\mathcal{A}_d| = |O| = \{0, 1, \ldots, 100\}$. In Figure 5.5, we present the optimal defense strategy against the insider with additivity given the standard deviation of the user's behavior. The strategies depict the defender's cumulative probability of setting a threshold at a certain value (for each threshold, the figure present the probability that this or any smaller threshold is selected). E.g., for std=1 the defender mixes only between the thresholds $a = 9$ and $a = 10$ (due to the allowed few false positives the defender play $a = 9$ with small probability). Indeed, in the case that the user's behavior is almost constant (normal distribution with std=1), the attacker cannot choose any strong action, since then the sum of the user's upload (which is almost certainly 10) and the attacker's action will almost certainly exceed the threshold and the attacker will be detected. However, in the case that the user's behavior is spread (e.g., std=19), the pure defense strategy is not effective as it would need to be set at quantile

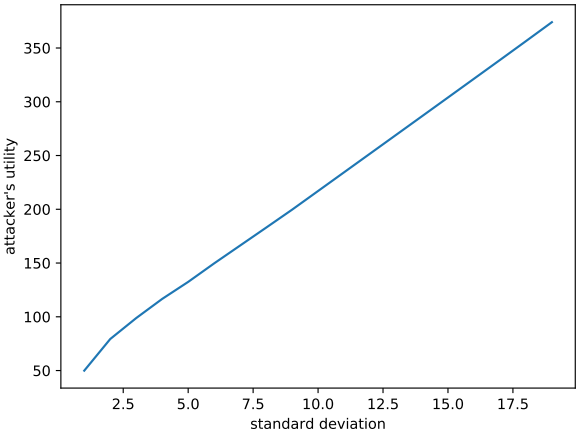


Figure 5.6: The attacker's expected utility given a user upload behavior follows normal distribution with mean equal to 10 and varying standard deviation (x-axis). It assumes the defender plays optimally according to Figure 5.5.

$1 - FP$. The optimal defender's strategy, as shown in Figure 5.5, mixes between multiple thresholds making the attacker indifferent across several attack actions (similarly, as the attacker's strategy against host type 3 in Figure 5.3b). However, all the attacker's actions yield him lower utility than what he would have had if the defender played different mixed or a pure strategy.

In Figure 5.6, we depict the attacker's utility for different standard deviations of the user's behavior. As the standard deviation of the user's behavior grows, the defense strategies are less effective and the attacker can exfiltrate more. Indeed, users with large uncertainty in their behavior are difficult to protect, since the thresholds for them must be set high not to create too many false positives. Which allows for the attacker to exfiltrate more easily without being detected.

5.6.5 Algorithm Scalability

In Figure 5.7 we present runtimes (note the logarithmic scale) for increasing numbers of considered host types. All experiments were run on an Intel Xeon E5-2650 2.6 GHz with time limit 2 hours. Strategies against the insider were computed in under 1 second, as they require single linear program computation. Adversarial HSVI, which iteratively improves the solution runs between ten seconds and two hours, depending on the parameters of the problem. Sometimes smaller problems took longer time to solve than the larger problems. E.g., the problem with only two host types took about 5 times longer than for seven host types. The reason is that the algorithm can temporarily stuck in sequences of solutions with no or very little improvements. The algorithm suffers with the plateaus during the computations even more than the original HSVI algorithm. See Figure 5.9, where we show the progress of the relative error ϵ in each iteration for the two

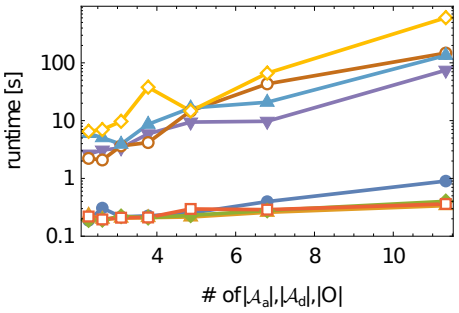


Figure 5.8: Algorithm runtimes for different sizes of set of actions $|\mathcal{A}_a|$, threshold $|\mathcal{A}_d|$ and observations $|O|$, with $|\mathcal{A}_d| = |\mathcal{A}_a| = |O|$. Legend is the same as in Figure 5.7.

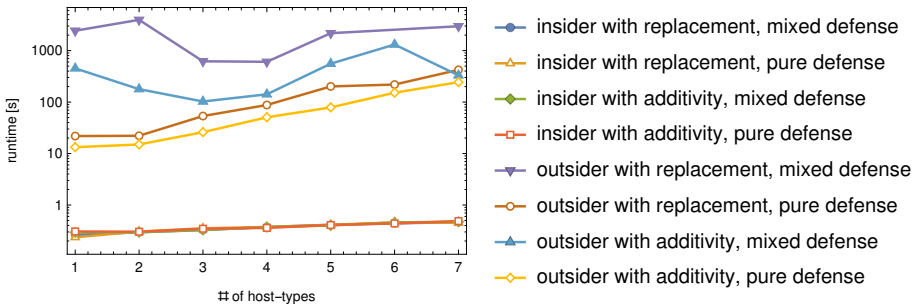


Figure 5.7: Algorithm runtimes for different number of host-types (clusters).

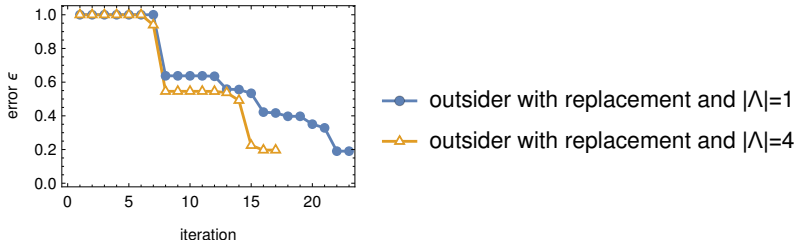


Figure 5.9: ϵ error progress during computation strategies for outsider with replacement with 1 and 4 number of types.

selected cases, with 1 and four host types, that demonstrates well the difference. Despite the fact that Adversarial HSVI with $|\mathcal{A}_c| = 4$ has 4 times more states, one can see that the algorithm was able to reach the desired error 20 in 17th iteration, while for $|\mathcal{A}_c| = 1$ at 25th iteration.

Finally, let analyze the algorithms scalability with the increasing number of actions, thresholds and observations. In Figure 5.8 (the legend is the same as in Figure 5.7) we plot the algorithm's runtime (note logarithmic y axis) for different number of the above-mentioned parameters (we plot only for the case that $|\mathcal{A}_d| = |\mathcal{A}_a| = |O|$). The algorithm was able to compute the problems of sizes $|\mathcal{A}_a| = |\mathcal{A}_d| = |O| = 15$, with error $\epsilon = 20\%$ and number of host types $|\mathcal{A}_c| = 7$ within 2 hours. This demonstrates that the algorithm can be used to solve practically large and real-world problems. In case of a need for faster algorithm, A-HSVI algorithm can be straightforwardly parallelize for concurrent computation of the algorithm.

5.7 Conclusion

In this chapter we demonstrated how our model and algorithms can be used to compute the thresholds for the detecting data exfiltration. We proposed two algorithms for computing the defender's strategies. Our first contribution, for the case that the attacker does not need to learn the behavior of the attacked host, is a linear programming formulation for computing the optimal strategies. This situation typically corresponds to insider attacks, such as employees, who know the standard behavior of the hosts in the network. Our second contribution, for the more complex situations, where the attacker learns the upload behavior of a user, is the Adversarial HSVI, an extension of the existing HSVI algorithm, to solve the zero-sum extensive-form game. Here, the attacker's strategies optimally weigh whether to attack aggressively from the beginning and risk detection or to carefully learn the host type from the observations and exfiltrate more afterward.

Using real-world user traffic, we validate that the proposed algorithms are sufficiently scalable to analyze realistic problems. The results of our case study show that richer models produce substantially better strategies. For example, when facing the external attacker that does not replace the original traffic, the simple heuristic defense strategies let the attacker exfiltrate three times more data than the strategy optimized against a perfectly informed

attacker using the linear program. Similarly, this strategy performs worse than the strategy optimized by Adversarial HSVI against the learning opponent. However, for too complex problems to be solved with Adversarial HSVI algorithm the simpler model, which assumes that the attacker knows the host types, can be used at cost of regret around 30%. Our results further show that randomized defense strategies can be up to 42% more effective in preventing data exfiltration compared to pure and deterministic strategies, which supports the same findings from [Lisý et al., 2014]. This is especially important when the attacker keeps the existing traffic intact, and the amounts of data transferred by the hosts vary substantially.

The attackers that know the exact behavior of the compromised host can exfiltrate by 14%–27% more data than the external attackers who have to learn it. Furthermore, a substantially more effective mitigation of data exfiltration is possible if the users are clustered into groups with similar behavior and a different detection strategy is used for each group. In our case study, the optimal strategy without the clustering allows the attacker to exfiltrate approximately three times more data than the optimal strategy using the clusters.

Our analysis showed, that in the case of the attackers with additivity the users with very unpredictable behavior are the most risky, since threshold for their hosts must be chosen high not to cause too many false positives. On a contrary, the users with behavior close to constant can be considered most safe.

Finally, we show that regardless of any other considered assumptions, if the attacker can replace the standard traffic of the compromised host, he can exfiltrate up to 6 times more data than the attacker who merely mixes his exfiltration traffic into the host's typical behavior. Therefore, monitoring the presence of the standard traffic (e.g., by expecting fake pre-scheduled transfers) may be a very effective countermeasure for decreasing the possible harm of data exfiltration.

Chapter 6

Conclusions

Since computer systems, deployed defenses, and attacks are becoming more complex, it is increasingly more difficult for the network administrators to assess the quality of the security countermeasures. It is difficult not only because of the enormous space of possible countermeasures that the administrator chooses from, but also because the attacker's reasoning process and his long-run adaptation to the countermeasures must be considered. Therefore, developing effective computer-aided configuration decision tools to improving security are essential. Game theory has already demonstrated its strategical strength in many specific games and physical infrastructures, including the cyber-security. Particularly in this field the strategic manipulation of information, deception tactics, and randomization are relevant.

This work was inspired and driven by the currently relevant decision-making problems in the network security area. In collaboration with the network security experts, we formalized two problems and developed game models for analyzing the interactions between the involved parties. In both of the presented models, the attacker's action space is enormous, and computing strategy profiles with the general state-of-the-art algorithms for imperfect information EFG with stochastic events is often intractable even for the toy problems. We overcame this problem by modeling the attacker's behavior using the POMDP and used its algorithms to compute the attacker's best response in combination with the game-theoretic algorithms. Whether the POMDP is finite-horizon or infinite-horizon often dictates what set of algorithms can be used to solve it. Since in each of the presented problem we used different POMDP to model the attacker's behavior, we had to take a different approach to solve the games. In the threshold selection problem, we modeled the attacker as a discounted infinite-horizon POMDP and developed a novel A-HSVI algorithm to compute the ϵ-Stackelberg equilibrium strategy profile. In the honeypot selection problem, the attacker's POMDP state-space is limited by the number of the actions in the attack graph; therefore, finite-horizon POMDP is used. We extended the forward dynamic programming with various pruning techniques, including the sibling-class theorem that exploits the attack graph structures. These techniques dramatically reduce the explored POMDP state-space to find the provably optimal attack policy. We showed that our solver significantly outperforms the unconstrained influence diagram approaches or other domain-independent solvers.

Computing the optimal defense strategies in our games had limited scalability. We explored how the alternation of the game assumptions impact the algorithmic aspect of the game by developing several heuristic approaches. In both problems, we showed that allowing the attacker to have slightly more information can result in the games solvable in polynomial time to the size of the game tree. E.g., in the honeypot selection problem, the attacker in the perfect-information game knew the number of deployed honeypots of each type. Although we called this heuristic perfect-information, the attacker had uncertainty whether a particular host of a given host type is a honeypot or a real host. Similarly, in the threshold selection problem the insider (with additivity) knew the upload probability distribution of the attacked host type but not the amount of data that host will upload in the next time step. Removing attacker's specific uncertainties allowed us computing the Stackelberg equilibrium of the altered game in polynomial time to the size of the game tree. Experimental evaluations of this heuristic showed that computed defense strategies have a low relative regret to the defender. Such findings would have been difficult to conclude without developing the optimal algorithms, presented in this work.

6.1 Thesis Contributions

This section summarizes the achievements and contributions of this thesis:

6.1.1 Game Models for Two Security Configuration Problem

One of our main contributions are novel game models for two current security configuration problems. Both security problems that we modeled in this thesis are still relevant for the network security community and have not been solved.

The first game model focuses on honeypot selection problem, where the defender selects configurations of the honeypots to deploy into the protected networks, and the attacker selects a contingency attack policy to perform in each of the observed configured networks. In [Durkota and Lisý, 2014] we published the model of the attacker that chooses an attack policy according to the attack graphs. The perfect-information game between the defender and the attacker was published in [Durkota et al., 2015b] and [Durkota et al., 2015c]. As far as we know, we are the first to use automatically generated complex attack graphs to model the attacker's choice of policies in combination with the game theory. Previous works used much simpler attack trees modeled by the administrators or experts. In [Durkota et al., 2015a] we extended the game to capture the attacker's prior belief about the possible systems.

The second model, published in [Durkota et al., 2017] focuses on the data exfiltration problem, where the defender chooses a data-upload threshold for each host type, and the attacker selects the amount of data to exfiltrate from each host type in different time steps. We analyzed two models of the attackers: an insider, who can replace data in the normal traffic, and an outsider, whose exfiltered data are added to the normal traffic of a user of the attacked host.

Finally, we presented a more general game model for security configuration problem that captures the essentials of this problem. Namely, the attacker's uncertainty about

the attacked system configurations and the defender's ability to influence the attacker's observations.

6.1.2 Set of Novel Algorithms

Our second contribution is developing a set of novel algorithms to solve the presented models. To solve honeypot selection problem, we had to develop variants of algorithms for computing the attacker's optimal attack policies over the attack graphs presented in [Durkota et al., 2015b] and [Durkota et al., 2015a]. Each algorithm assumes a different degree of the attacker's certainty about the attacked network. We proposed and experimentally evaluated several pruning techniques that allow the algorithms to scale to larger problems. Algorithms that find exact SSE do not scale well, we further developed error-bounded MILP and set of heuristic algorithms. The heuristic algorithms introduce additional assumptions that allow solving the problem in polynomial time in the size of the game tree.

For threshold selection problem, we proposed a linear program to compute optimal defense strategy against the insider's attack model. To compute the defense strategy against the outsider's attack model we proposed Adversarial HSVI algorithm to include the defender's decisions.

6.1.3 Experimental Evaluations and Analyses

Computing the optimal defense strategies is crucial for measuring and evaluating the quality of the error-bounded and heuristic approach approaches. We carried out the extensive experimental evaluations of the algorithms in terms of their running times and the quality and robustness of computed strategies. For the experiments we used real-world data in threshold selection problem and network topologies inspired by the real-world network topologies in the honeypot selection problem. We demonstrated how game-theory could be used to answer various questions, such as, what happens if certain model assumptions are not met, or what is the defender's regret for incorrectly assuming the attacker's model (motivation, costs, etc.). In [Durkota et al., 2016] we compared game-theoretic strategies to human strategies and analyze their shortcomings. Answering these question gave us a better insight into the proposed strategies.

6.1.4 Implementation

The described algorithms and problem representations were implemented in game-theoretic library *GTLibrary* and can be reused in future to solve other problems, or to as a basis for other novel algorithms.

6.2 Future Work

Our work has potential for a further research in several directions.

Scalable Algorithms Our current algorithms for the honeypot selection problem are still not scalable enough to be used for large-scale networks. Developing additional clever

grouping of the hosts or more efficient algorithms and heuristics to solve these problems is important for the model to be applicable in the networks that are increasingly more complex.

Extending Defender's Actions The model for honeypot selection problem currently allows only one type of countermeasures for the defender, deploying honeypots. The model could be extended to allow other actions, such as firewall configurations, configuration of the available services, etc. However, it is useful to model only those countermeasures that have an impact on the attack graphs of the networks.

Generalizing Threshold Selection Model The threshold selection model can be significantly generalized to capture more complex features for the detector and various goals for the attacker. E.g., game could be played on a multi-featured space of the user's traffic behavior. The defender selects a decision boundaries on the feature space (like thresholds) for the host to be inspected. The attacker on the other hand chooses what action to perform based on their rewards and how the action influences the features observed by the detector. Such generalized game can be used to model different problems of detecting malicious behavior.

Real-world Tests The presented models are tested in a simulated environment on real-world inspired scenarios. We hope to apply the computed strategies in real-world cases, where the real attackers would attempt to compromise the networks or exfiltrate the data. This would give us a valuable feedback how well the strategies perform and what are their potential drawbacks.

Exploiting Payoff Structure Our game has a specific payoff structure consisting of the general components as follows: (i) reward/loss, which is a zero-sum component after a constant multiplication, (ii) attacker's action costs, (iii) defender's action costs. We believe, that deeper theoretical analysis of this payoff structure could lead to developing more efficient algorithms, that exploited the payoff structure.

Multiple Defenders Lastly, defining an extended version of strong Stackelberg equilibrium for more than one leader/defender, that would always exist and be stable belongs to open problems and is of great interest to many game-theory researchers. This solution concept could have a great real-world impact on problems, where multiple defenders cooperate against a single opponent to profit as much as possible. In domains, such as physical security, network security, or cyber-security, this solution concept could be well applied.

6.3 Publications

In this section, we list all publications of the author, structured into related and unrelated topics to this thesis.

6.3.1 Publications Related to the Thesis

Journal Publications (with IF) (2)

- Durkota, K., Lisý, V., Kiekintveld, C., Bošanský B. and Pěchouček, M. (2016). Case studies of network defense with attack graph games. *IEEE Intelligent Systems*, 31(5):24–30 (**2 citations, IF=3.532 as of 2016**, 45% contribution)

Conference Publications (5)

- Durkota, K., Lisý, V., Kiekintveld, C., Horá, K., Bošanský, B., and Pevný, T. (2017). Optimal strategies for detecting data exfiltration by internal and external attackers. *In International Conference on Decision and Game Theory for Security*, pages 171–192. Springer (**1 citations**, 16.67%)

- Durkota, K., Lisý, V., Bošanský, B., and Kiekintveld, C. (2015). Approximate solutions for attack graph games with imperfect information. *In International Conference on Decision and Game Theory for Security*, pages 228–249. Springer (**6 citations**, 38% contribution)

- Durkota, K., Lisý, V., Bošanský B., and Kiekintveld, C. (2015). Optimal network security hardening using attack graph games. *In Proceedings of IJCAI*, pages 7–14 (**14 citations**, 38% contribution, Rank: A*)

- Durkota, K., Lisý, V., Kiekintveld, C., and Bošanský, B. (2015). Game-theoretic algorithms for optimal network security hardening using attack graphs. *In Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 1773–1774. International Foundation for Autonomous Agents and Multiagent Systems (**6 citations**, 38% contribution, Rank: A*)

- Durkota, K. and Lisý V. (2014). Computing optimal policies for attack graphs with action failures and costs. *In STAIRS*, pages 101–110 (**5 citations**, 70% contribution)

6.3.2 Publications Not Related with the Thesis

Conference Publications (6)

- Tožička, J., Jakubuv, J., Durkota, K., and Komenda, A. (2015). Extensibility based multiagent planner with plan diversity metrics. *In Transactions on Computational Collective Intelligence XX*, pages 117–139. Springer (**0 citations**, 25% contribution)

- Čermák J., Bošanský, B., Durkota, K., Lisý V., and Kiekintveld, C. (2016). Using correlated strategies for computing Stackelberg equilibria in extensive-form games. *In Thirtieth AAAI Conference on Artificial Intelligence* (**0 citations**, 15% contribution)

- Tožička, J., Jakubuv, J., Durkota, K., Komenda, A., and Pěchouček, M. (2014). Multiagent planning supported by plan diversity metrics and landmark actions.*In ICAART*, pages 178–189 (**1 citations**, 30% contribution)

- Tožička, J., Jakubuv, J., Durkota, K., and Komenda, A. (2014). Multiagent planning by iterative negotiation over distributed planning graphs. *In Proceedings of the Workshop on Distributed and Mutiltiagent Planning (ICAPS)*, pages 7–15 (**1 citations**, 30% contribution)

- Durkota, K. and Komenda, A. (2013). Deterministic multiagent planning techniques: experimental comparison (short paper). *In Proceedings of DMAP workshop of ICAPS*, volume 13, pages 43–47 (**4 citations**, 85% contribution)

- Marr, J., Pere, A., Durkota, K., Rogers, A., Fish, V., and Arndt, M. (2011a). Laboratory exercises using the haystack VSRT interferometer to teach the basics of aperture synthesis. *arXiv preprint arXiv:1109.3816* (**0 citations**)

Bibliography

[mca, 2015] (2015). Grand theft data data exfiltration study: Actors, tactics, and detection. techreport, McAfee, Inc.

[Albanese et al., 2016] Albanese, M., Battista, E., and Jajodia, S. (2016). Deceiving attackers by creating a virtual attack surface. In *Cyber Deception*, pages 169–201. Springer.

[Almeshekah and Spafford, 2016] Almeshekah, M. H. and Spafford, E. H. (2016). Cyber security deception. In *Cyber Deception*, pages 25–52. Springer.

[Ammann et al., 2002] Ammann, P., Wijesekera, D., and Kaushik, S. (2002). Scalable, graph-based network vulnerability analysis. In *CCS*, pages 217–224.

[Astrom, 1965] Astrom, K. J. (1965). Optimal control of markov decision processes with incomplete state estimation. *Journal of mathematical analysis and applications*, 10:174–205.

[Bacic et al., 2006] Bacic, E., Froh, M., and Henderson, G. (2006). Mulval extensions for dynamic asset protection. Technical report, DTIC Document.

[Barik et al., 2016] Barik, M. S., Sengupta, A., and Mazumdar, C. (2016). Attack graph generation and analysis techniques. *Defence Science Journal*, 66(6):559.

[Bellman, 1956] Bellman, R. (1956). Dynamic programming and lagrange multipliers. *PNAS*, 42(10):767.

[Bellovin and Bush, 2009] Bellovin, S. M. and Bush, R. (2009). Configuration management and security. *IEEE Journal on Selected Areas in Communications*, 27(3):268–274.

[Benisch et al., 2010] Benisch, M., Davis, G. B., and Sandholm, T. (2010). Algorithms for closed under rational behavior (curb) sets. *Journal of Artificial Intelligence Research*, pages 513–534.

[Bertsekas et al., 1995] Bertsekas, D. P., Bertsekas, D. P., Bertsekas, D. P., and Bertsekas, D. P. (1995). *Dynamic programming and optimal control*, volume 1. Athena scientific Belmont, MA.

[Bi et al., 2016] Bi, K., Han, D., and Wang, J. (2016). K maximum probability attack paths dynamic generation algorithm. *Computer Science and Information Systems*, 13(2):677–689.

[Bistarelli et al., 2006a] Bistarelli, S., Dall'Aglio, M., and Peretti, P. (2006a). Strategic games on defense trees. In *International Workshop on Formal Aspects in Security and Trust*, pages 1–15. Springer.

[Bistarelli et al., 2006b] Bistarelli, S., Fioravanti, F., and Peretti, P. (2006b). Defense trees for economic evaluation of security investments. In *Availability, Reliability and Security, 2006. ARES 2006. The First International Conference on*, pages 8–pp. IEEE.

[Boddy et al., 2005] Boddy, M. S., Gohde, J., Haigh, T., and Harp, S. A. (2005). Course of action generation for cyber security using classical planning. In *ICAPS*, pages 12–21.

[Borbor et al., 2017] Borbor, D., Wang, L., Jajodia, S., and Singhal, A. (2017). Securing networks against unpatchable and unknown vulnerabilities using heterogeneous hardening options. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 509–528. Springer.

[Bošanský et al., 2014] Bošanský, B., Kiekintveld, C., Lisý, V., and Pěchouček, M. (2014). An exact double-oracle algorithm for zero-sum extensive-form games with imperfect information. *Journal of Artificial Intelligence Research*, 51:829–866.

[Bošanský and Čermák, 2015] Bošanský, B. and Čermák, J. (2015). Sequence-form algorithm for computing stackelberg equilibria in extensive-form games. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 805–811. AAAI Press.

[Bowen et al., 2009] Bowen, B. M., Hershkop, S., Keromytis, A. D., and Stolfo, S. J. (2009). Baiting inside attackers using decoy documents. In *International Conference on Security and Privacy in Communication Systems*, pages 51–70. Springer.

[Braziunas, 2003] Braziunas, D. (2003). Pomdp solution methods. *University of Toronto, Tech. Rep.*

[Buldas and Stepanenko, 2012] Buldas, A. and Stepanenko, R. (2012). *Upper Bounds for Adversaries' Utility in Attack Trees*, pages 98–117. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Carroll and Grosu, 2011] Carroll, T. E. and Grosu, D. (2011). A game theoretic investigation of deception in network security. *Security and Communication Networks*, 4(10):1162–1172.

[Cassandra et al., 1997] Cassandra, A., Littman, M. L., and Zhang, N. L. (1997). Incremental pruning: A simple, fast, exact method for partially observable markov decision processes. In *Proceedings of the Thirteenth conference on Uncertainty in artificial intelligence*, pages 54–61. Morgan Kaufmann Publishers Inc.

[Cassandra, 1998] Cassandra, A. R. (1998). A survey of pomdp applications. In *Working notes of AAAI 1998 fall symposium on planning with partially observable Markov decision processes*, volume 1724.

[Cheng and Wellman, 2007] Cheng, S.-F. and Wellman, M. P. (2007). Iterated weaker-than-weak dominance. In *IJCAI*, pages 1233–1238.

[Cheng et al., 2014] Cheng, Y., Deng, J., Li, J., DeLoach, S. A., Singhal, A., and Ou, X. (2014). Metrics of security. In *Cyber Defense and Situational Awareness*, pages 263–295. Springer.

[Comesana et al., 2006] Comesana, P., Pérez-Freire, L., and Pérez-González, F. (2006). Blind newton sensitivity attack. *IEE Proceedings-Information Security*, 153(3):115–125.

[Conitzer and Korzhyk, 2011] Conitzer, V. and Korzhyk, D. (2011). Commitment to correlated strategies. In *AAAI*.

[Conitzer and Sandholm, 2006] Conitzer, V. and Sandholm, T. (2006). Computing the optimal strategy to commit to. In *EC*, pages 82–90.

[Couch, 2008] Couch, A. L. (2008). System configuration management. In *Handbook of Network and System Administration*, pages 75–133. Elsevier.

[Durkota et al., 2016] Durkota, K., Kiekintveld, C., Pechoucek, M., et al. (2016). Case studies of network defense with attack graph games. *IEEE Intelligent Systems*, 31(5):24–30.

[Durkota and Lisý, 2014] Durkota, K. and Lisý, V. (2014). Computing optimal policies for attack graphs with action failures and costs. In *STAIRS*, pages 101–110.

[Durkota et al., 2015a] Durkota, K., Lisý, V., Bošanský, B., and Kiekintveld, C. (2015a). Approximate solutions for attack graph games with imperfect information. In *Decision and Game Theory for Security*, pages 228–249. Springer.

[Durkota et al., 2015b] Durkota, K., Lisý, V., Bošanský, B., and Kiekintveld, C. (2015b). Optimal network security hardening using attack graph games. In *Proceedings of IJCAI*, pages 7–14.

[Durkota et al., 2015c] Durkota, K., Lisy, V., Kiekintveld, C., and Bo˘ sanský, B. (2015c). Game-theoretic algorithms for optimal network security hardening using attack graphs. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 1773–1774. International Foundation for Autonomous Agents and Multiagent Systems.

[Durkota et al., 2017] Durkota, K., Lisỳ, V., Kiekintveld, C., Horák, K., Bošanskỳ, B., and Pevnỳ, T. (2017). Optimal strategies for detecting data exfiltration by internal and external attackers. In *International Conference on Decision and Game Theory for Security*, pages 171–192. Springer.

[Fadolalkarim et al., 2016] Fadolalkarim, D., Sallam, A., and Bertino, E. (2016). Pandde: Provenance-based anomaly detection of data exfiltration. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, pages 267–276. ACM.

[Fang et al., 2017] Fang, F., Nguyen, T. H., Pickles, R., Lam, W. Y., Clements, G. R., An, B., Singh, A., Schwedock, B. C., Tambe, M., and Lemieux, A. (2017). Paws-a deployed game-theoretic application to combat poaching. *AI Magazine*, 38(1):23–36.

[Feder et al., 2007] Feder, T., Nazerzadeh, H., and Saberi, A. (2007). Approximating nash equilibria using small-support strategies. In *Proceedings of the 8th ACM conference on Electronic commerce*, pages 352–354. ACM.

[Fitzgerald et al., 2013] Fitzgerald, W. M., Neville, U., and Foley, S. N. (2013). Automated smartphone security configuration. In *Data Privacy Management and Autonomous Spontaneous Security*, pages 227–242. Springer.

[Garg and Grosu, 2007] Garg, N. and Grosu, D. (2007). Deception in honeynets: A game-theoretic analysis. In *Information Assurance and Security Workshop, 2007. IAW'07. IEEE SMC*, pages 107–113. IEEE.

[Greiner et al., 2006] Greiner, R., Hayward, R., Jankowska, M., and Molloy, M. (2006). Finding optimal satisficing strategies for and-or trees. *Artificial Intelligence*, pages 19–58.

[Grimes, 2005] Grimes, R. A. (2005). *Honeypots for Windows*. Books for Professionals by Professionals. Springer, Dordrecht.

[Harsanyi, 1967] Harsanyi, J. C. (1967). Games with incomplete information played by "bayesian" players, i–iii part i. the basic model. *Management science*, 14(3):159–182.

[Hauskrecht, 1997] Hauskrecht, M. (1997). Incremental methods for computing bounds in partially observable markov decision processes. In *AAAI/IAAI*, pages 734–739. Citeseer.

[Hayatle et al., 2012] Hayatle, O., Otrok, H., and Youssef, A. (2012). A game theoretic investigation for high interaction honeypots. In *Communications (ICC), 2012 IEEE International Conference on*, pages 6662–6667. IEEE.

[Hoey et al., 1999] Hoey, J., St-Aubin, R., Hu, A., and Boutilier, C. (1999). Spudd: Stochastic planning using decision diagrams. In *UAI*, pages 279–288.

[Homer et al., 2013] Homer, J., Zhang, S., Ou, X., Schmidt, D., Du, Y., Rajagopalan, S. R., and Singhal, A. (2013). Aggregating vulnerability metrics in enterprise networks using attack graphs. *Journal of Computer Security*, pages 561–597.

[Horák and Bošanský, 2016] Horák, K. and Bošanský, B. (2016). A point-based approximate algorithm for one-sided partially observable pursuit-evasion games. In *International Conference on Decision and Game Theory for Security*, pages 435–454. Springer.

[Hu et al., 2017] Hu, H., Zhang, H., Liu, Y., and Wang, Y. (2017). Quantitative method for network security situation based on attack prediction. *Security and Communication Networks*, 2017.

[Ingols et al., 2006] Ingols, K., Lippmann, R., and Piwowarski, K. (2006). Practical attack graph generation for network defense. In *Proceedings of the 22nd Annual Computer Security Applications Conference*, pages 121–130. IEEE Computer Society.

[Isa et al., 2007] Isa, J., Lisy, V., Reitermanova, Z., and Sykora, O. (2007). Unconstrained influence diagram solver: Guido. In *IEEE ICTAI*, pages 24–27.

[Jajodia et al., 2005] Jajodia, S., Noel, S., and O'Berry, B. (2005). Topological analysis of network attack vulnerability. In *Managing Cyber Threats*, pages 247–266. Springer.

[Jajodia et al., 2017] Jajodia, S., Park, N., Pierazzi, F., Pugliese, A., Serra, E., Simari, G. I., and Subrahmanian, V. (2017). A probabilistic logic of cyber deception. *IEEE Transactions on Information Forensics and Security*, 12(11):2532–2544.

[Jha et al., 2002] Jha, S., Sheyner, O., and Wing, J. (2002). Two formal analyses of attack graphs. In *Computer Security Foundations Workshop, 2002. Proceedings. 15th IEEE*, pages 49–63. IEEE.

[JIA and Zhi-Chang, 2010] JIA, Y. Y. X. X.-S. and Zhi-Chang, Y. Q. (2010). An attack graph-based probabilistic computing approach of network security [j]. *Chinese Journal of Computers*, 10:021.

[Kaelbling et al., 1998] Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134.

[Kalai and Lehrer, 1993] Kalai, E. and Lehrer, E. (1993). Rational learning leads to nash equilibrium. *Econometrica: Journal of the Econometric Society*, pages 1019–1045.

[Kearns et al., 2001] Kearns, M., Littman, M. L., and Singh, S. (2001). Graphical models for game theory. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pages 253–260. Morgan Kaufmann Publishers Inc.

[Koller et al., 1996] Koller, D., Megiddo, N., and Von Stengel, B. (1996). Efficient computation of equilibria for extensive two-person games. *Games and economic behavior*, 14(2):247–259.

[Korzhyk et al., 2011] Korzhyk, D., Yin, Z., Kiekintveld, C., Conitzer, V., and Tambe, M. (2011). Stackelberg vs. nash in security games: An extended investigation of interchangeability, equivalence, and uniqueness. *Journal of Artificial Intelligence Research*, 41(2):297–327.

[Krautsevich et al., 2012] Krautsevich, L., Martinelli, F., and Yautsiukhin, A. (2012). Towards modelling adaptive attacker's behaviour. In *International Symposium on Foundations and Practice of Security*, pages 357–364. Springer.

[Laszka et al., 2016a] Laszka, A., Abbas, W., Sastry, S. S., Vorobeychik, Y., and Koutsoukos, X. (2016a). Optimal thresholds for intrusion detection systems. In *Proceedings of the Symposium and Bootcamp on the Science of Security*, pages 72–81. ACM.

[Laszka et al., 2016b] Laszka, A., Lou, J., and Vorobeychik, Y. (2016b). Multi-defender strategic filtering against spear-phishing attacks.

[Lee et al., 2002] Lee, S. Y., Low, W. L., and Wong, P. Y. (2002). Learning fingerprints for a database intrusion detection system. In *European Symposium on Research in Computer Security*, pages 264–279. Springer.

[Letchford and Conitzer, 2010] Letchford, J. and Conitzer, V. (2010). Computing optimal strategies to commit to in extensive-form games. In *Proceedings of the 11th ACM conference on Electronic commerce*, pages 83–92. ACM.

[Letchford and Vorobeychik, 2013] Letchford, J. and Vorobeychik, Y. (2013). Optimal interdiction of attack plans. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 199–206. International Foundation for Autonomous Agents and Multiagent Systems.

[Lippmann et al., 2002] Lippmann, R. et al. (2002). Netspa: A network security planning architecture. *Massachusetts Institute of Technology*.

[Lisý et al., 2014] Lisý, V., Kessl, R., and Pevný, T. (2014). Randomized operating point selection in adversarial classification. In *Machine Learning and Knowledge Discovery in Databases*, pages 240–255. Springer Berlin Heidelberg.

[Lisý and Píbil, 2013] Lisý, V. and Píbil, R. (2013). Computing optimal attack strategies using unconstrained influence diagrams. In *PAISI*, pages 38–46.

[Liu et al., 2008] Liu, D., Wang, X., and Camp, J. (2008). Game-theoretic modeling and analysis of insider threats. *International Journal of Critical Infrastructure Protection*, 1:75–80.

[Liu and Kuhn, 2010] Liu, S. and Kuhn, R. (2010). Data loss prevention. *IT professional*, 12(2).

[Liu et al., 2009] Liu, Y., Corbett, C., Chiang, K., Archibald, R., Mukherjee, B., and Ghosal, D. (2009). Sidd: A framework for detecting sensitive data exfiltration by an insider attack. In *System Sciences, 2009. HICSS'09. 42nd Hawaii International Conference on*, pages 1–10. IEEE.

[Lou et al., 2015] Lou, J., Smith, A. M., and Vorobeychik, Y. (2015). Multidefender security games. *arXiv preprint arXiv:1505.07548*.

[Lou and Vorobeychik, 2015] Lou, J. and Vorobeychik, Y. (2015). Equilibrium analysis of multi-defender security games. In *Proceedings of the 24th International Conference on Artificial Intelligence*, pages 596–602. AAAI Press.

[Mc Carthy et al., 2016] Mc Carthy, S. M., Sinha, A., Tambe, M., and Manadhata, P. (2016). Data exfiltration detection and prevention: Virtually distributed pomdps for practically safer networks. In *International Conference on Decision and Game Theory for Security*, pages 39–61. Springer.

[McMahan et al., 2003] McMahan, H. B., Gordon, G. J., and Blum, A. (2003). 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.

[Mell et al., 2006] Mell, P., Scarfone, K., and Romanosky, S. (2006). Common vulnerability scoring system. *Security & Privacy*, pages 85–89.

[Mitchell and Healy, 2018] Mitchell, R. and Healy, B. (2018). A game theoretic model of computer network exploitation campaigns. In *Computing and Communication Workshop and Conference (CCWC), 2018 IEEE 8th Annual*, pages 431–438. IEEE.

[Moravčík et al., 2017] Moravčík, M., Schmid, M., Burch, N., Lisỳ, V., Morrill, D., Bard, N., Davis, T., Waugh, K., Johanson, M., and Bowling, M. (2017). Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356(6337):508–513.

[Nandi et al., 2016] Nandi, A. K., Medal, H. R., and Vadlamani, S. (2016). Interdicting attack graphs to protect organizations from cyber attacks: A bi-level defender–attacker model. *Computers & Operations Research*, 75:118–131.

[Noel and Jajodia, 2004] Noel, S. and Jajodia, S. (2004). Managing attack graph complexity through visual hierarchical aggregation. In *Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, pages 109–118. ACM.

[Noel and Jajodia, 2008] Noel, S. and Jajodia, S. (2008). Optimal ids sensor placement and alert prioritization using attack graphs. *Journal of Network and Systems Management*, pages 259–275.

[Noel et al., 2010] Noel, S., Jajodia, S., Wang, L., and Singhal, A. (2010). Measuring security risk of networks using attack graphs. *International Journal of Next-Generation Computing*, 1(1):135–147.

[Obes et al., 2013] Obes, J. L., Sarraute, C., and Richarte, G. (2013). Attack planning in the real world. *arXiv preprint arXiv:1306.4044*.

[Ou et al., 2006] Ou, X., Boyer, W. F., and McQueen, M. A. (2006). A scalable approach to attack graph generation. In *CCS*, pages 336–345.

[Ou et al., 2005] Ou, X., Govindavajhala, S., and Appel, A. W. (2005). Mulval: A logic-based network security analyzer. In *USENIX Security*.

[Ou and Singhal, 2012] Ou, X. and Singhal, A. (2012). *Quantitative Security Risk Assessment of Enterprise Networks*. SpringerBriefs in Computer Science. Springer-Verlag New York, 1 edition.

[Peterside et al., 2015] Peterside, G. B., Zavarsky, P., and Butakov, S. (2015). Automated security configuration checklist for a cisco ipsec vpn router using scap 1.2. In *Internet Technology and Secured Transactions (ICITST), 2015 10th International Conference for*, pages 355–360. IEEE.

[Píbil et al., 2012] Píbil, R., Lisý, V., Kiekintveld, C., Bošanský, B., and Pěchouček, M. (2012). Game theoretic model of strategic honeypot selection in computer networks. In *Decision and Game Theory for Security*, pages 201–220. Springer.

[Pita et al., 2008] Pita, J., Jain, M., Marecki, J., Ordóñez, F., Portway, C., Tambe, M., Western, C., Paruchuri, P., and Kraus, S. (2008). Deployed armor protection: the application of a game theoretic model for security at the los angeles international airport. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: industrial track*, pages 125–132. International Foundation for Autonomous Agents and Multiagent Systems.

[Polatidis et al., 2017] Polatidis, N., Pavlidis, M., and Mouratidis, H. (2017). Cyber-attack path discovery in a dynamic supply chain maritime risk management system. *Computer Standards & Interfaces*.

[Provos, 2004] Provos, N. (2004). A virtual honeypot framework. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 1–1, Berkeley, CA, USA. USENIX Association.

[Qassrawi and Hongli, 2010] Qassrawi, M. T. and Hongli, Z. (2010). Deception methodology in virtual honeypots. In *Networks Security Wireless Communications and Trusted Computing (NSWCTC), 2010 Second International Conference on*, volume 2, pages 462–467. IEEE.

[Ritchey and Ammann, 2000] Ritchey, R. W. and Ammann, P. (2000). Using model checking to analyze network vulnerabilities. In *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*, pages 156–165. IEEE.

[Rubner et al., 2000] Rubner, Y., Tomasi, C., and Guibas, L. J. (2000). The earth mover's distance as a metric for image retrieval. *International journal of computer vision*, 40(2):99–121.

[Sarraute et al., 2012] Sarraute, C., Buffet, O., Hoffmann, J., et al. (2012). Pomdps make better hackers: Accounting for uncertainty in penetration testing. In *AAAI*.

[Sawilla and Ou, 2008] Sawilla, R. E. and Ou, X. (2008). Identifying critical attack assets in dependency attack graphs. In Jajodia, S. and Lopez, J., editors, *Computer Security - ESORICS 2008*, volume 5283 of *Lecture Notes in Computer Science*, pages 18–34. Springer Berlin Heidelberg.

[Schiffman et al., 2004] Schiffman, M., Wright, A., Ahmad, D., and Eschelbeck, G. (2004). The common vulnerability scoring system. *National Infrastructure Advisory Council, Vulnerability Disclosure Working Group, Vulnerability Scoring Subgroup*.

[Schwartz, 1970] Schwartz, T. (1970). On the possibility of rational policy evaluation. *Theory and Decision*, 1(1):89–106.

[Sheyner et al., 2002] Sheyner, O., Haines, J., Jha, S., Lippmann, R., and Wing, J. (2002). Automated generation and analysis of attack graphs. In *IEEE S&P*, pages 273–284.

[Shoham and Leyton-Brown, 2008] Shoham, Y. and Leyton-Brown, K. (2008). *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press.

[Silver et al., 2017] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017). Mastering the game of go without human knowledge. *Nature*, 550(7676):354.

[Smith et al., 2014] Smith, A., Vorobeychik, Y., and Letchford, J. (2014). Multidefender security games on networks. *ACM SIGMETRICS Performance Evaluation Review*, 41(4):4–7.

[Smith and Simmons, 2004] Smith, T. and Simmons, R. (2004). Heuristic search value iteration for pomdps. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 520–527. AUAI Press.

[Sommestad and Sandström, 2015] Sommestad, T. and Sandström, F. (2015). An empirical test of the accuracy of an attack graph analysis tool. *Information & Computer Security*, 23(5):516–531.

[Speicher et al., 2018] Speicher, P., Steinmetz, M., and Backes, M. (2018). Stackelberg planning: Towards effective leader-follower state space search.

[Spitzner, 2003a] Spitzner, L. (2003a). The honeynet project: Trapping the hackers. *IEEE Security & Privacy*, 99(2):15–23.

[Spitzner, 2003b] Spitzner, L. (2003b). Honeypots: Catching the insider threat. In *Computer Security Applications Conference, 2003. Proceedings. 19th Annual*, pages 170–179. IEEE.

[Straffin, 1993] Straffin, P. D. (1993). *Game theory and strategy*, volume 36. MAA.

[Tambe, 2011] Tambe, M. (2011). *Security and Game Theory: Algorithms, Deployed Systems, Lessons Learned*. Cambridge University Press.

[Tsai et al., 2009] Tsai, J., Kiekintveld, C., Ordonez, F., Tambe, M., and Rathi, S. (2009). Iris-a tool for strategic security allocation in transportation networks.

[Vaněk et al., 2010] Vaněk, O., Bošanskỳ, B., Jakob, M., and Pěchouček, M. (2010). Transiting areas patrolled by a mobile adversary. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 9–16. IEEE.

[Čermák et al., 2016a] Čermák, J., Bošanský, B., Durkota, K., Lisý, V., and Kiekintveld, C. (2016a). Using correlated strategies for computing stackelberg equilibria in extensive-form games. In *Thirtieth AAAI Conference on Artificial Intelligence*.

[Čermák et al., 2016b] Čermák, J., Bošanský, B., Durkota, K., Lisý, V., and Kiekintveld, C. (2016b). Using correlated strategies for computing stackelberg equilibria in extensive-form games. In *Thirtieth AAAI Conference on Artificial Intelligence*.

[Von Neumann and Morgenstern, 2007] Von Neumann, J. and Morgenstern, O. (2007). *Theory of games and economic behavior*. Princeton university press.

[Von Stackelberg, 2010] Von Stackelberg, H. (2010). *Market structure and equilibrium*. Springer Science & Business Media.

[von Stengel and Forges, 2008] von Stengel, B. and Forges, F. (2008). Extensive-form correlated equilibrium: Definition and computational complexity. *Mathematics of Operations Research*, 33(4):1002–1022.

[von Stengel and Zamir, 2010] von Stengel, B. and Zamir, S. (2010). Leadership games with convex strategy sets. *Games and Economic Behavior*, 69(2):446–457.

[Wang et al., 2008a] Wang, L., Islam, T., Long, T., Singhal, A., and Jajodia, S. (2008a). An attack graph-based probabilistic security metric. In Atluri, V., editor, *Data and Applications Security XXII*, volume 5094 of *Lecture Notes in Computer Science*, pages 283–296. Springer Berlin Heidelberg.

[Wang et al., 2008b] Wang, L., Islam, T., Long, T., Singhal, A., and Jajodia, S. (2008b). An attack graph-based probabilistic security metric. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 283–296. Springer.

[Zander et al., 2007] Zander, S., Armitage, G., and Branch, P. (2007). A survey of covert channels and countermeasures in computer network protocols. *IEEE Communications Surveys & Tutorials*, 9(3):44–57.

Appendix A

Notation

A.0.1 List of Notations

In Table A.1 we provide a list of notations we use in the thesis.

Table A.1: List of used notation and functions.

(PO)MDP	
S	set of states
A	set of actions
$P[s'\|a,s]$	probability of reaching state s' from s if a is performed
$R(s,a,s')$	reward for performing action a in s leading to s'
$\pi : S \to A$	MDP policy
$V^{\pi}(s)$	value of policy π starting in state s
O	set of observations in POMDP
$O[o\|s',a]$	probability of observing o if action a is played and state s' is reached
γ	discount factor
$\mathcal{B} = \Delta(S)$	probability distribution (belief space) over states S
$\kappa(I) \subseteq \Xi(I)$	is set of strategies in SUS for I
$\pi : \mathcal{B} \to A$	POMDP policy
Game Model	
$\mathcal{N}$	finite set of players
$\mathcal{A}_i$	finite set of player i's actions in a game
u_i	player i's utility function
Π_i	set of all pure strategies of player i
Δ_i	set of all mixed strategies of player i
$\mathcal{H}$	finite set of nodes in the EFG
$\mathcal{Z} \subseteq \mathcal{H}$	finite set of terminal nodes in EFG
$\mathcal{P} : \mathcal{H} \to \mathcal{N} \cup \{c\}$	a function that assigns player to takes action in a given node, where c denote a Chance player
$\mathcal{C} : \mathcal{H} \to [0,1]$	denotes the probability of reaching node h due to Chance
$\mathcal{I}$ (resp. $\mathcal{I}_i$)	set of all (resp. player i) information sets in a game

Σ_i	denotes all sequences of player i
$\mathrm{BR}_i(\sigma_{-i})$	function, that returns set of best-response strategies of player i against strategy σ_{-i}
$g_i : \Sigma_i \times \Sigma_{-i} \to \mathbb{R}$	an extended payoff function for all $\mathcal{H}$
$\mathrm{seq}_i(I_i)$	denotes a sequence σ_i for player i that leads to I_i
$A_i(I_i)$	set of player i's actions in $I_i \in \mathcal{I}_i$
Attack Graph	
$AG(n)$	the attack graph of network s
T	the set of host types in attack graph
F	the set of facts in attack graph
A	the set of atomic actions in attack graph
p_a	success probability of action a in an attack graph
c_a	the attacker's action cost in an attack graph
τ_a	the set of host types that action a interacts with
r_f	attacker's reward for achieving fact f
$\mathit{eff}(a)$	the set of effects of action a
$\mathit{pre}(a)$	the set of preconditions of action a
Attack Policy	
ξ	is an attack policy
V	vertices in an attack policy
E, E^+, E^-	all, positive and negative edges in an attack policy, respectively
$\mathit{path}(\xi, v_l)$	path in attack policy ξ from root to vertex v_l (excluding)

A.0.2 List of Abbreviations

In Table A.2 we provide a list abbreviations used in this thesis.

Table A.2: List of used abbreviations

AG	attack graph
SUS	smallest undominated set
EFG	extensive-Form Game
FDP	forward dynamic programming
FP	false-positive
GS	general-sum
HSP	Honeypot Selection Problem
(A-)HSVI	(Adversarial) Heuristic Search Value Iteration
LB	lower bound
(PO)MDP	(Partially Observable) Markov Decision Process
NE	Nash equilibrium
NFG	normal-Form Game
(S)SE	(strong) Stackelberg equilibrium
UB	upper bound
ZS	zero-sum