Bachelor Project

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Computing Defense Strategies in Security Games with Unknown Actions of the Attacker

David Čech

Supervisor: doc. Mgr. Branislav Bošanský, Ph.D.
May 2023

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Čech David**	Personal ID number:	**499129**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Cybernetics**		
Study program:	**Open Informatics**		
Specialisation:	**Artificial Intelligence and Computer Science**		

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Computing Defense Strategies in Security Games with Unknown Actions of the Attacker

Bachelor’s thesis title in Czech:

Strategie obránce v bezpečnostních hrách s neznámými akcemi útočníka

Guidelines:

A common assumption in game theory is that both players know the rules of the game and all executable actions. However, in practical security problems, the attacker can have actions that are not known to the defender (e.g., zero-day exploits).
The goal of the student is to:
(1) Formulate game-theoretic models for a network-security problem where the attacker has a richer set of actions than the defender assumes (zero-day exploits).
(2) Implement this type of game in the OpenSpiel framework.
(3) Evaluate the robustness of different game-theoretic strategies computed on a game without additional actions (zero-day exploits) when used on instances of the game with additional actions. Evaluate different solution concepts as well as different algorithms.
(4) Implement and evaluate the impact of an online adaptation of the defender’s game-theoretic strategies when used in the game with additional actions.

Bibliography / sources:

[1] Abu Sayed, Ahmed Anwar, Christopher Kiekintveld, Branislav Bosansky, and Charles Kamhoua "Cyber Deception against Zero-day Attacks: A Game Theoretic Approach.“ International Conference on Decision and Game Theory for Security. 2022
[2] Mar Lanctot et al. "OpenSpiel: A framework for reinforcement learning in games." arXiv preprint arXiv:1908.09453 (2019).
[3] Shoham, Yoav, and Leyton-Brown, Kevin. "Multiagent systems." Cambridge Books (2009).

Name and workplace of bachelor’s thesis supervisor:

doc. Mgr. Branislav Bošanský, Ph.D. Artificial Intelligence Center FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **12.01.2023** Deadline for bachelor thesis submission: **26.05.2023**

Assignment valid until: **22.09.2024**

doc. Mgr. Branislav Bošanský, Ph.D.	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Acknowledgements

I would like to express my deep gratitude to my advisor, doc. Mgr. Branislav Bošanský, Ph.D., for his invaluable guidance and support throughout the completion of this work. His expertise and encouragement helped me on many occasions and ultimately enabled me to complete my assignment.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, May 26, 2023
David Čech

Abstract

In this work, we examine a problem faced by computer network administrators defending their designated networks against adversarial attacks. We assume that the defender is able to hinder the attacker's progress by deploying deceptive services called honeypots. Our goal is to find an optimal allocation of honeypots that maximizes the resources required by the attacker to infiltrate a key host of the computer network. However, a major complication in this scenario is that the defender might be aware only of a limited amount of services that might be exploited by the attacker in the target network. We use game theory to model this scenario where the attacker has actions unknown to the defender and to find optimal strategies in such a setting. We compare the results obtained in this domain with the results obtained in a game where both the defender and the attacker have the same amount of information about the vulnerabilities in the computer network. We investigate how the game dynamics change between these two games in relation to the size of the game and compare the strength of the defender's policies received from different algorithms. We point out problems that arise when the defender deploys a policy computed using their limited information. We propose several ways of increasing the strength of the defender's strategy and highlight their limitations. Finally, we explore two of these proposed methods and present the acquired results.

Keywords: game theory, network security, dynamic honeypot allocation, machine learning, imperfect information games

Supervisor: doc. Mgr. Branislav Bošanský, Ph.D.
Department of Computer Science, Czech Technical University in Prague

Abstrakt

V této práci zkoumáme problém, se kterým se potýkají administrátoři počítačových sítí při snaze chránit svou síť před nepřátelskými útoky. Předpokládáme, že má obránce sítě možnost bránit útočníkově postupu pomocí falešných služeb zvaných honeypoty. Našim cílem je nalézt strategii rozmístění honeypotů, která maximalizuje využití útočníkových prostředků k infiltraci kritické infrastruktury. Velkým problémem však je, že si obránce nemusí být vědom všech služeb, které útočník může využít k pohybu v počítačové síti. Využíváme teorii her k modelování tohoto scénaře, v němž má útočník akce, o kterých obránce neví, a k nalezení optimálních strategií pro oba hráče. Porovnáváme výsledky obdržené na této doméně s výsledky obdrženými na doméně, kde mají oba hráči stejnou informaci o struktuře počítačové sítě. Zkoumáme změnu herní dynamiky těchto dvou her v závislosti na jejich velikosti a porovnáváme sílu strategií, jež jsou výstupy různých algoritmů. Upozorňujeme na problémy, které vznikají, při aplikaci strategie spočítané obráncem na základě jeho omezených informací. Navrhujeme několik způsobů, jak zvýšit sílu obráncovy strategie, a poukazujeme na jejich limitace. V poslední řadě zkoušíme dva postupy pro zlepšení obráncovy strategie a prezentujeme získané výsledky.

Klíčová slova: teorie her, síťové zabezpečení, dynamická alokace honeypotů, strojové učení, hry s neúplnými informacemi

Překlad názvu: Strategie obránce v bezpečnostních hrách s neznámými akcemi útočníka

Contents

Figures

Tables

Chapter 1
Introduction

The domain of computer network security is an example of a conflict between two rational agents with opposing interests. With the rise of the use of computer networks in the military (Xin and Bin, 2013), industry (Christin et al., 2010), autonomous vehicles (Hussain and Zeadally, 2019), and many other areas of life, there has also been an increase in attempts to compromise these networks and to gain access to valuable information or to control the systems mentioned above (Bhushan and Sahoo, 2017). In response to these attacks, there has also been an increasing effort to prevent these malicious attackers from achieving their goals. One of the ways a computer network administrator can hinder the attacker's advances is to introduce false vulnerabilities into the network called **honeypots**. If the attacker attempts to utilize these false vulnerabilities, they use their resources (e.g. computational time) and they do not advance in the target network. Honeypots may also log the attacker's activity, thus informing the defender about the attacker's progress in the network. The honeypot technology as well as honeypot-related software and analysis of recorded logs are described in detail in Nawrocki et al., 2016. However, there are two main limitations to the use of honeypots. First, it can be quite costly to set up services acting as false vulnerabilities, especially if they are to deceive the attacker, therefore the defender usually has only a limited number of honeypots at their disposal. Second, the defender can set up these services only if they know about the possibility of the attacker exploiting the real vulnerabilities. It is, however, possible that the defender is unaware of some vulnerabilities in the network making it impossible to set up honeypots for these **hidden vulnerabilities** (e.g. zero-day exploits). The primary goal of this work is to produce a strategy for the placement of honeypots which maximizes the attacker's use of resources in a computer network, where hidden vulnerabilities may exist.

The problem of finding an optimal strategy in a scenario, where two rational agents have opposing interests, is thoroughly studied in the field of game theory. Game theory allows for reasoning in domains, where there are multiple rational agents having various degrees of information and varying interests. However, one of the common assumptions in game theory is that the actions of the individual agents are well-known. Therefore game theory does not provide a simple solution to our problem with hidden vulnerabilities, where a subset of the attacker's actions is hidden from the defender. Nevertheless, we use game theory to formally define our problem and use existing solution concepts to find strategies for the defender. We evaluate these strategies against various opponents and explore ways to improve the defender's expected utility. Since exact mathematical approaches have only a limited degree of scalability, we also employ reinforcement-learning approaches for finding optimal strategies in our domain. Additionally, we showcase problems arising from the complexity of the space of all possible defender strategies. We consider a case in which both players are unaware of the policy of the other player and a case in which the attacker knows the defender's strategy in advance.

In this work we present three main contributions. First, we present an implementation of our domain in the OpenSpiel framework (Lanctot et al., 2019) which enables the use of existing algorithms on our domain. Second, we conduct experiments to ascertain whether there are major differences in the quality of solutions provided by the individual algorithms. Third, we suggest three ways in which the found solutions might be improved and try two of these approaches. Subsequently, we present the results obtained by our novel approaches and compare them with the solutions acquired by traditional methods.

The structure of this work is as follows. This chapter serves as an introduction to the researched topic. In Chapter 2, we present basic game theoretic definitions and solution concepts. Then in Chapter 3, we formally define the **lateral movement** domain and describe its implementation in OpenSpiel. Subsequently, in Chapter 4, we introduce algorithms used to find solutions and provide reasoning behind their choice. Next, in Chapter 5, we describe conducted experiments and provide their results. Finally, in Chapter 6 we summarise our findings and conclude this work. Additionally, Appendix A presents the bibliography of sources for this work. Moreover, in Appendix B, can be found implementation-specific experimental settings. And lastly, Appendix C describes the structure of the project.

Chapter 2

Introduction to Game Theory

Since this project is very closely related to game theory, this chapter serves as a brief introduction to basic game-theoretic definitions and solution concepts utilized in subsequent chapters. All of the following definitions are excerpts from Shoham and Leyton-Brown, 2008. Game theory itself is the mathematical study of interaction among independent, self-interested agents. It has been applied in a number of fields including economics, psychology, and computer science.

2.1 Types of Games

The most important definition for our problem is the definition of a **perfect-information game in extensive form**.

Definition (Perfect-information game) A (finite) perfect-information game (in extensive form) is a tuple $G = (N, A, H, Z, \chi, \rho, \sigma, u)$, where:

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

The term of **utility function** deserves further explanation. Utility functions are used to describe the interests of the individual agents in games. In our case, the utility functions are mappings from terminal nodes (i.e. outcomes of a game) to real numbers, which express the agent's subjective preference for a particular outcome as opposed to the other outcomes. The extensive form enables us to keep the representation of our game relatively concise compared to other available game models such as normal-form games. Additionally, the extensive form accounts for the fact that players might take turns in choosing actions and that the game might consist of multiple turns that happen one after the other. In addition to the extensive form, we also need to represent the fact that the players are not always aware of the actions of the other player. To this end serves the concept of **imperfect-information** games.

Definition (Imperfect-information game) An imperfect-information game (in extensive form) is a tuple $G = (N, A, H, Z, \chi, \rho, \sigma, u, I)$, where:

- $G = (N, A, H, Z, \chi, \rho, \sigma, u, I)$ is a perfect-information extensive-form game; and
- $I = (I_1, ..., I_n)$, where $I_i = (I_{i,1}, ..., I_{i,k_i})$ is a set of equivalence classes on (i.e., a partition of) $\{h \in H : \rho(h) = i\}$ with the property that $\chi(h) = \chi(h')$ and $\rho(h) = \rho(h')$ whenever there exists a j for which $h \in I_{i,j}$ and $h' \in I_{i,j}$

In imperfect-information extensive form games the choice nodes are partitioned into information sets. An information set includes all choice nodes that appear identical to the player because the states of the game that correspond to these choice nodes differ only in the part of the game which is hidden from the player. The lack of information about the state of the game may also come from the fact that the player does not have a complete record of their own actions, or of the observed actions of other players. This concept is explored in games of **perfect recall**.

Definition (Perfect recall) Player i has perfect recall in an imperfect-information game G if for any two nodes h, h' that are in the same information set for player i, for any path $h_0, a_0, h_1, a_1, h_2, ..., h_m, a_m, h$ from the root of the game to h (where the h_j are decision nodes and the a_j are actions) and for any path $h_0, a'_0, h'_1, a'_1, h'_2, ..., h'_m, a'_m, h'$ from the root to h' it must be the case that:

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

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

If any of the conditions in the definition above does not hold for any player of the game, the game is of **imperfect recall**. Additionally, all games of perfect-information are by definition games of perfect recall. In our domain, we assume both the attacker and the defender to have perfect recall. Another important concept in game theory is the degree of cooperation of players in a game. In the general case player's utility might not be related to the utility of any other player, however, there are scenarios, where the interests of players are directly opposed. This is modeled by **zero-sum games**.

Definition (Zero-sum game) A two-player extensive form game is zero-sum, if for all $z \in Z, u_1(z) = -u_2(z)$.

The definition of a zero-sum game represents the fact that the gain of one player inevitably causes harm to the other player.

2.2 Strategies

One of the goals of game theory is to be able to identify optimal behavior in various games. However, there is no simple answer to the question of what optimal behavior in games is. Game theory provides multiple solution concepts, many of which can be further expanded to find optimal solutions under slightly different criteria. A solution of a game is usually given in the form of a strategy that satisfies specified optimality criteria. In an extensive form imperfect-information game strategies are defined upon the individual information sets. **Pure strategies** are a category of strategies that specify in each information set one specific action to be played.

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

However, in imperfect-information games, it is often beneficial to randomize over the set of all available actions according to a given probability distribution, because it makes the player hard to predict. There are two categories of strategies that deal with randomization over available actions. One of these categories contains **mixed strategies** which randomize over pure strategies. The other consists of **behavioral strategies** which randomize independently over available actions at each information set. The difference is that when adhering to a mixed strategy the player decides at the beginning according to a given probability distribution which pure strategy they are going to play for the rest of the game, whereas in the case of adhering to a behavioral strategy, the player repeatedly randomizes throughout the course of the game each time they encounter a choice node according to a probability distribution of its information set. A tuple $s = (s_1, ..., s_n)$ composed of strategies of the respective players is called a **strategy profile**.

2.3 Solution Concepts

The notion of **best response** is a stepping stone for arguably the most important solution concept of the **Nash equilibrium**.

Definition (Best response) Player i' best response to the strategy profile s_{-i} is a strategy $s_i^* \in S_i$ such that $u_i(s_i^*, s_{-i}) \geq u_i(s_i, s_{-i})$ for all strategies $s_i \in S_i$, where s_{-i} denotes strategy profile of all players except the player i.

In other words, best response is a strategy that maximizes the player i's utility when they play with players who follow strategies known to the player i beforehand.

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

One of the reasons why Nash equilibrium is such an important solution concept is that it is a stable strategy profile because any deviation from it by any player would decrease their utility. Other reasons for its significance entail the fact that every game has at least one Nash equilibrium (Nash's theorem), and that some other solution concepts in special cases coincide with Nash equilibria, such as the minmax, and maxmin strategies in two-player zero-sum games (Minimax theorem). A minor modification to the concept of Nash equilibrium provides **ϵ-Nash equilibrium** which reflects the fact that computation of an exact Nash equilibrium is in most domains infeasible, and as such a formal definition of an approximate Nash equilibrium is required.

Definition (ϵ-Nash equilibrium) Fix $\epsilon > 0$. A strategy profile $s = (s_1, ..., s_n)$ is an ϵ-Nash equilibrium if, for all agents i and for for all strategies $s_i' \neq s_i$, $u_i(s_i, s_{-i}) \geq u_i(s_i', s_{-i}) - \epsilon$.

2.4 Limitations of game theory

Even though algorithms for computing optimal solutions to the criteria mentioned above exist, in practice the domains are often extensive to a degree that an exact solution becomes intractable. Moreover, if the game is happening in real-time, and the domain changes dynamically, the solutions would need to be computed anew after every change in the environment. These two reasons motivate the application of reinforcement learning algorithms to game-theoretic problems in areas, where traditional solution methods face difficulties.

Chapter 3
Lateral Movement

3.1 Description

The **lateral movement game** described in Horák et al., 2019, and Sayed et al., 2023 represents a scenario in cyber security featuring two players: an **attacker** and a **defender**. The goal of the attacker is to reach a specific host in the network, such as the central database, using vulnerabilities in the network. The defender aims to hinder the attacker's progress by deploying decoys, called **honeypots**, which expend the attacker's resources and alert the defender about the attacker's progress in the network. To model the computer networks with vulnerabilities we use directed acyclic graphs. An example of an abstract computer network represented by a graph can be seen in Figure 3.1. This graph serves as a leading example for most of the results presented in this work.

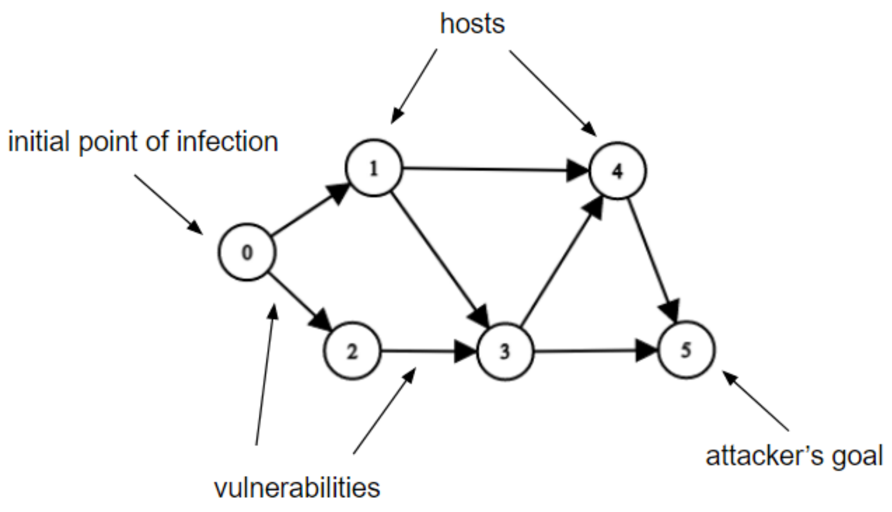

Figure 3.1: Example network represented by a graph

We assume that there initially exists a singular host in the network through which the attacker can enter the network and that the initial point of infection is well-known. The game consists of a specified number of stages. Each stage begins with the defender choosing a small number of services to honeypot (i.e. a subset of edges in the graph). Then the attacker chooses which vulnerabilities in the network they exploit to compromise additional hosts (edges leading from infected nodes). The attacker may do so until they reach their goal or until they trigger a honeypot. Upon triggering a honeypot, the game progresses into a new stage and the defender may choose a new set of services to honeypot. The game ends either when the attacker reaches their goal or if a specified number of stages elapses. We introduce the notion of vulnerabilities **hidden** from the defender. These hidden vulnerabilities represent the fact that the defender might not be aware of all the vulnerabilities that exist in the computer network and thus is unable to allocate honeypots to these hidden vulnerabilities. The attacker has full information about the vulnerabilities in the network and thus may utilize the hidden vulnerabilities to infiltrate hosts. We also introduce a version of the game, where the honeypots have a given probability of defect. If the attacker uses a honeypot and it defects, the attacker continues to infiltrate the computer network uninterrupted and the defender receives no information about the attacker's progress. We do this to investigate the properties of more robust strategies learned by the defender in such games.

From a game-theoretic standpoint the game is zero-sum because the players have directly opposing interests, as well as imperfect-information because the individual players are not aware of what the actions of the other player are. Furthermore, we model the game as an extensive form game, where players take turns in a sequential manner. We assume the games to be of perfect recall, imposing no limits to the amount of information the individual players can remember. However, we break the property of perfect recall in the information set representation used by reinforcement learning agents in favor of decreasing the size of the state space and faster convergence rate of the algorithms. This is possible due to the fact that multiagent reinforcement learning algorithms do not rely on the property of perfect recall, unlike some of the strictly mathematical approaches. There are chance nodes in the lateral movement game with defect that determine the outcome of the attacker's use of a honeypot. If the probability of defect is 0, the game is without chance. Lastly, the players receive their respective utilities whenever a terminal state is reached.

3.2 Formal Game Definition

Formally the scenario described in the previous section can be modeled by a directed acyclic graph $G = (N, E)$, where the set of nodes N represents hosts of the computer network, and the set of directed edges E represents the vulnerabilities. We define sets E_h and E_k which stand for vulnerabilities hidden from the defender and vulnerabilities known to the defender, respectively. Set

E is partitioned into E_h, and E_k, therefore $E = E_h \cup E_k$, and $E_h \cap E_k = \emptyset$. There are costs associated with the edges, which are expressed by the function $f_c : E \to \mathbb{R}^+$. Moreover, each edge is also assigned to a security perimeter tied to a multiplicative constant which models the attacker's penalization upon using a honeypot. This is denoted by $f_p : E \to \mathbb{R}^+$. Furthermore, let $P = \{1, 2\}$ denote the set of players, where 1 stands for the defender, and 2 stands for the attacker. Each stage begins with the defender allocating k honeypots to arbitrary known edges of the graph. The set of honeypots at stage s is denoted $H_s \subseteq E_k, |H_s| = k$. After the placement of the honeypots, the attacker takes their turn. Let I_t denote a set of nodes infected by the attacker at step t. We define $I_0 = \{n_0\}$, where n_0 is the initial well-known point of infection. At step $t \geq 1$ the attacker may choose to infect any node n_j for which there exists an edge $e_{ij} \in E$ leading from node $n_i \in I_{t-1}$ to n_j. If $e_{ij} \notin H_s$, where s stands for the current stage, then $I_t = I_{t-1} \cup \{n_j\}$, and the attacker chooses another node to infect at step $t + 1$ in the same way. If $e_{ij} \in H_s$, then in the attacker does not infect n_j, thus $I_t = I_{t-1}$, both players receive observations about the attacker's use of the honeypot, and the game either progresses into a new stage or terminates if the stage number reaches a terminal value. If there is defect present in the game, whenever the attacker chooses $e_{ij} \in H_s$, a random chance outcome is sampled according to a probability distribution $p(d) = q$, and $p(\overline{d}) = 1 - q$, where d denotes the defect of a honeypot, and $\overline{d}$ denotes the standard behavior. If the honeypot malfunctions, then $I_t = I_{t-1} \cup \{n_j\}$, the stage remains the same, no observations are delivered to the players and the attacker proceeds to infect another node. Conversely, if the honeypot functions properly the game dynamics are the same as in a situation where the attacker uses a honeypot in the game without defect described above.

Let $E_s = (e_1, ..., e_n)$ denote the sequence of edges used by the attacker in stage s. If stage s was terminated by the attacker triggering a honeypot, then the defender's reward for the stage is $R_1^s = \sum_{i=1}^{n-1} f_c(e_i) + f_c(e_n) \cdot f_p(e_n)$. If stage s is instead terminated by the attacker reaching the terminal node, then the defender's reward is $R_1^s = \sum_{i=1}^{n} f_c(e_i) - c$, where c is a positive constant representing the attacker's reward for reaching the terminal node. Since the game is zero-sum, the attacker's reward is always equal to the negative value of the defender's reward $R_2^s = -R_1^s$. The terminal reward for player i is then $R_i = \sum_{s \in S} R_i^s$ normalized to interval $[-1, 1]$ (S denotes the set of all stages). The goal of the players is to maximize their respective terminal rewards.

3.3 Graph Generation

The implementation of the lateral movement game described in this work contains a random generator for the host/vulnerability graphs, as well as tools for specifying the hyperparameters of the game. The random generation is taken from Horák et al., 2019, and is described in detail in Algorithm 1. However, some parts of the algorithm need further explanation. The nodes are sorted at line 6 in ascending order with regard to their Euclidean distance

Algorithm 1 Graph generation

```
1:  $\boldsymbol{n}_0 \leftarrow (0,0)$
2:  $\boldsymbol{n}_n \leftarrow (20,20)$
3:  for $i$ in $1 \ldots n-1$ do
4:      $\boldsymbol{n}_i \leftarrow \text{rand}([0,20]^2)$
5:  end for
6:  $N \leftarrow \textbf{sort}(\boldsymbol{n}_0, \ldots, \boldsymbol{n}_n)$
7:  for $i$ in $0 \ldots n-1$ do
8:      $\text{Neigh}_i = \textbf{FindNearestNeighbors}(\boldsymbol{n}_i, d)$
9:  end for
10: $E \leftarrow \{\}$
11: for $i$ in $0 \ldots n-1$ do
12:     for $j$ in $0 \ldots d-1$ do
13:         $x \leftarrow \text{Neigh}_i\,[j]$
14:         if $\textbf{EdgeDoesNotExist}(i,x)$ then
15:             $E \leftarrow E \cup \textbf{CreateEdge}(i,x)$
16:         end if
17:     end for
18: end for
19: $E_h \leftarrow \textbf{SelectHiddenEdges}(E)$
20: $E_k \leftarrow E \setminus E_h$
21: $G \leftarrow (N, E_k \cup E_h)$
22: if $G$ is not valid then
23:     go to 3
24: end if
```

from $\boldsymbol{n}_0$. Euclidean distance is also used for the purpose of finding nearest neighbors at line 8. An edge may be chosen to be a hidden edge only if the origin of the edge has an out-degree greater than 1, and the end has an in-degree greater than 1. A graph is considered an invalid graph if the amount of edges that satisfy this condition is lesser than the value of the number of hidden edges hyperparameter. Similarly, if the graph contains a node with a 0 in-degree, or 0 out-degree, it is also considered invalid. The only exceptions to this rule are $\boldsymbol{n}_0$ and $\boldsymbol{n}_n$ which always have 0 in-degree and 0 out-degree, respectively. Lastly, if the graph is too sparse, it is too considered invalid. The generation of the graph is repeated until either a valid graph is generated or a specified amount of invalid graphs has been generated, in which case the generation terminates with an error. A possible layout of the nodes generated by the algorithm can be seen in Figure 3.2. The figure depicts the leading example that can be seen in Figure 3.1, where the layout of nodes was altered to make the figure less cluttered.

For a graph generated by the above-described algorithm, costs for edges are computed using several functions. The function $f(e_{ij}) = \|\boldsymbol{n}_i - \boldsymbol{n}_j\|_2 \int_0^1 g(\lambda \boldsymbol{n}_i + (1-\lambda)\boldsymbol{n}_j)\, d\lambda$, where $g(x,y) = x + y$, has many desirable properties for f_c, but the values obtained by this function can be quite large. Therefore the output of f is normalized to the interval $[0, 3]$, which is denoted by f_{norm}. Final cost of an edge is then $f_c(e_{ij}) = (f_{norm} \circ f)(e_{ij})$. Additionally, three security perimeters for nodes are defined based on their Euclidean distance from $\boldsymbol{n}_n$. Edges always inherit their security perimeter from the node they lead into. The security perimeters are depicted in Figure 3.2 by the two dashed circles. Subsequently, the generation of costs and assignment of security perimeters can be seen in Figure 3.3. Each edge is labeled by its id and its cost. The edge colors correspond to their security perimeter assignments. We use the distance thresholds for the security parameters equal to 12, and 18 for security perimeters $(2, 3)$ and $(1, 2)$, respectively. The security perimeters are associated with multiplicative constants $(\kappa_1, \kappa_2, \kappa_3) = (1, 1.5, 4)$. We denote this mapping $f_\kappa : N \to \{\kappa_1, \kappa_2, \kappa_3\}$. The final penalization function for honeypotted edges is then $f_p(e_{ij}) = f_\kappa(\boldsymbol{n}_j)$.

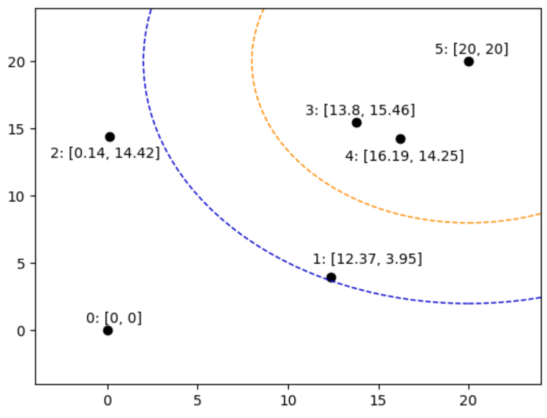

Figure 3.2: Nodes generated by Algorithm 1

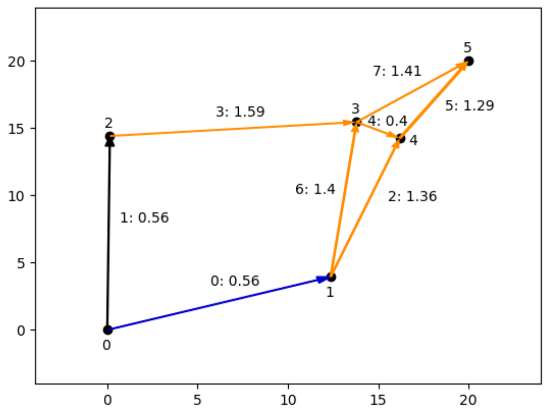

Figure 3.3: Edges generated by Algorithm 1

3.4 Game Parametrization

User may specify hyperparameters for both the generation of the graph and the game itself. The implementation provides the following hyperparameters:

- number of honeypots
- seed
- density
- number of nodes
- number of stages
- number of hidden edges
- enable hidden edges
- defect rate

The number of nodes and density define n and d in Algorithm 1, respectively. The number of honeypots corresponds to k in the formal game definition and determines how many honeypots the defender is able to place in each stage of the game. Subsequently, the number of stages defines the maximum number of stages played before the game terminates. The game may terminate in an earlier stage if the attacker reaches the terminal node. Additionally, the seed specifies the random number generator seed for the graph generation. Furthermore, the number of hidden edges controls $|E_h|$. An important thing to note is that the hidden edges are always generated regardless of whether they are used in the game or not. Whether or not the attacker can use the generated hidden edges for the infection of nodes is determined by the enable hidden edges parameter. Lastly, defect rate defines the probability of a honeypot defect, i.e. $p(d)$.

3.5 Information Sets

The OpenSpiel framework offers two primary forms of representation of the information sets a given state of the game belongs to. The first of these forms is an **information state string**. The information state string is an arbitrary identifier of an information set used mainly by algorithms that perform computations on the entire game trees. The information state string in our game satisfies a property of perfect recall that there is a single sequence of player's actions that leads into a given information set. This is required by some of the algorithms utilized to find optimal strategies. The other representation is that of an **information state tensor**. Information state tensor is a representation used primarily by agents utilizing neural networks. As such the information state tensor is an array of floating point numbers that is used directly as an input to the input layer of the neural networks. Therefore it is desirable to represent the information set in the most compact way possible. Since perfect recall representation is not necessary for the reinforcement-learning agents, we give up the perfect recall property for the information state tensors by grouping states of the game which are results of sequences of actions in a different order but are otherwise equivalent. This helps the deep learning agents to converge faster to equilibrium strategies in our experimental evaluation.

The information state string contains the following information: the current stage of the game, a player id, known actions played by the attacker in each stage, and known actions played by the defender in each stage. These fields are delimited by the pipe symbol, and actions played in different stages of the game are delimited by a semicolon. In the information state tensor, however, the representation is not as straightforward. The stage number and player identifier both use one-hot encoding. Moreover, the tensor contains one bit for each node in the graph and one bit for each known edge per stage of the game. Value 0 of these bits signifies that the node is not infected or that a honeypot was not placed on the given edge in a particular stage. Conversely, value 1 means that the node is infected or that there was a honeypot placed on that edge in the given stage.

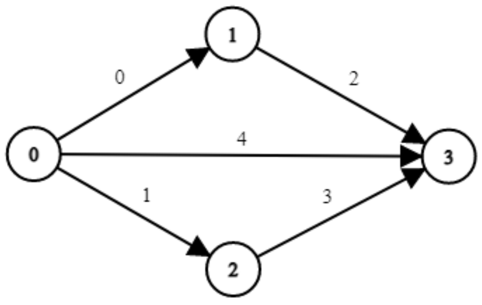

Figure 3.4: Information set example graph

Let us see an example of the information set generation on the graph depicted in Figure 3.4. Consider the following sequence of actions: first, the defender places honeypots on edges 2 and 4, then the attacker infects node 1 via edge 0 (i.e. $e_{0,1}$) and tries to infect node 3 via edge 4 triggering the honeypot and progressing the game into a new stage. In the second stage (numbered stage 1, since numbering starts at 0), the defender places honeypots on edges 3 and 1 and the attacker infects node 3 via edge 2 reaching their goal and thus terminating the game. These actions are not part of an optimal strategy and serve only as an example for the generation of the information set identifiers. The defender's information state string can be seen in Figure 3.5. Similarly, the attacker's information state string can be seen in Figure 3.6. Additionally, the information tensors for the defender and the attacker are depicted in Figure 3.7 and Figure 3.8, respectively.

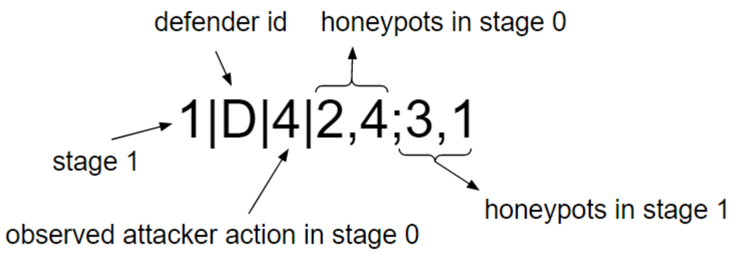

Figure 3.5: Defender's information state string

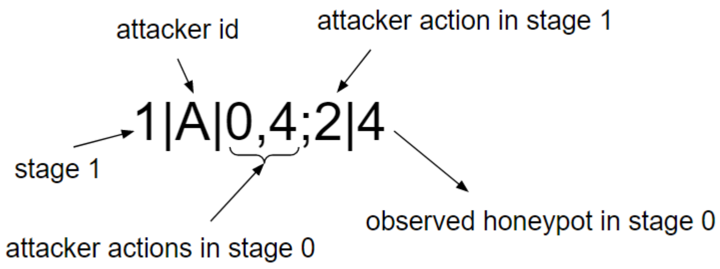
attacker id
attacker action in stage 1
1|A|0,4;2|4
stage 1
attacker actions in stage 0
observed honeypot in stage 0

Figure 3.6: Attacker's information state sting

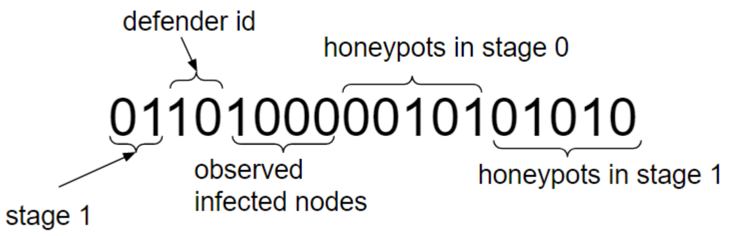

Figure 3.7: Defender's information state tensor

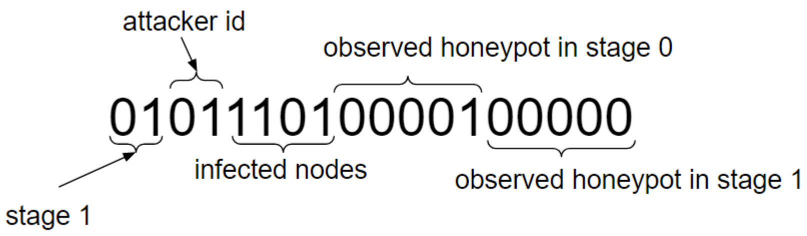

Figure 3.8: Attacker's information state tensor

A couple of things to note: the defender does not know about the attacker having infected node 1, only about the infection of the node from which the spotted attack originates i.e. node 0, but has perfect information about their honeypot placement, which the attacker lacks. Additionally, if the defender placed their honeypots in a different order, the information state tensors would stay the same, whereas the information state strings would differ.

3.6 OpenSpiel

In this section, we provide a brief description of the OpenSpiel framework and the reasoning behind our use of OpenSpiel. OpenSpiel introduced in Lanctot et al., 2019 is an open-source collection of environments and algorithms created and maintained by DeepMind. Its purpose is to facilitate research of reinforcement learning and search/planning in games. The environments are developed mainly in `C++`, and exposed to a `Python` API with the use of `pybind11`. The majority of algorithms and examples are then implemented in `Python` using this API.

There are several benefits to utilizing OpenSpiel for custom game environments. OpenSpiel provides a unified interface through which many baseline algorithms for planning in games, as well as reinforcement learning algorithms, can be applied to custom domains. Furthermore, many of these baseline algorithms have already been tested, and their functionality has been verified. The full list of algorithms, and their testing status is available in the official OpenSpiel documentation. Additionally, the OpenSpiel framework accounts for many game-theoretic concepts, such as player strategies, simultaneous turn games, extensive form games, games with perfect, and imperfect information, games with perfect, and imperfect recall, and chance elements in games. Moreover, OpenSpiel provides a way to include hyperparameters into the game model. Another benefit to using OpenSpiel is that it comes with tests for correct integration of the unified game interface into the custom domains and scripts that simplify compilation, building, and testing of the entire project. Lastly, OpenSpiel contains tools for the evaluation of strategies in the form of expected player utilities and convergence to a Nash equilibrium.

The files created for the purposes of this project are described in Appendix C. When compiling from source, the default procedure for the building of OpenSpiel described in Lanctot et al., 2019 compiles the lateral movement domain as well. It also runs the lateral movement tests by default. The `Python` scripts require the installation of requirements included in the project and the setting of `PATH` and `PYTHONPATH` which is too described in the mentioned OpenSpiel article.

Chapter 4

Algorithms

4.1 Motivation

When deciding what algorithms to use for our experiments, we wanted to include both state-of-the-art and baseline algorithms implemented in OpenSpiel. We chose algorithms denoted as thoroughly-tested in the OpenSpiel documentation to minimize the risk of encountering incorrect behavior. Among the baseline algorithms, we chose sequence-form linear programming (SFLP, Koller et al., 1994) and counterfactual regret minimization (CFR, Zinkevich et al., 2007). In terms of state-of-the-art algorithms, we selected deep counterfactual regret minimization (Brown et al., 2019), exploitability descent (ED, Lockhart et al., 2020), and neural fictitious self-play (NFSP, Heinrich and Silver, 2016). The thought process behind the selection was to use the baseline algorithms on small game trees to validate the implementation of our domain in the OpenSpiel framework, as well as the function of the more complex state-of-the-art algorithms on our domain, then scale up the size of the game and drop algorithms infeasible on larger domains. Since multiple Nash equilibria often exist in nontrivial domains, we also wanted to compare the strategies found by the algorithms to ascertain whether some of the algorithms find strategies yielding on average higher utilities for the defender than others.

During the experimental evaluation, however, we discovered that the execution time of ED was 25 times as long as that of NFSP, presumably due to the way loss is calculated on the entire game trees in each iteration, making the runtime at least 17 hours on medium domains, where exact methods are still feasible. Because of that, we decided to exclude ED from our experiments. Similarly, the execution of deep CFR was 15 times slower than that of NFSP on medium-sized domains making the runtime about 7 hours. These runtime estimates have been computed for the number of iterations used on trivial domains and it is possible that the number of iterations would need to be larger in order to obtain sufficient results on medium-sized domains which would inflate these figures further. Additionally, in large domains, a single iteration of deep CFR did not terminate in 10 minutes making the minimum estimate of runtime in large domains about 14 days and possibly much longer. Since the primary motivation behind using deep CFR was its scalability, we also decided against using it in our experiments. Due to these reasons only SFLP, CFR, and NFSP are discussed in more detail in this work.

4.2 Sequence-Form Linear Programming

Sequence-form linear programming utilizes the fact that the problem of finding a Nash equilibrium in a two-player perfect recall zero-sum game can be expressed by a linear program. This is due to the fact that the Nash equilibrium is in such games equivalent to a pair of max-min strategies. The classic linear program formulation is defined for a normal-form game which is represented by a payoff matrix for each possible strategy combination of the two players. The solution of the linear program can then be found in polynomial time in the size of the payoff matrix. The problem with this approach is, however, that the size of the payoff matrix is exponential in the size of the game tree representation making it infeasible for any practical problems.

The above-mentioned issue is addressed by Koller et al., 1994 in creating a linear program equivalent to the normal-form problem formulation but decreasing its size to only linear in the size of the game tree. This decrease in size is due to defining strategies on the **sequence form**, instead of converting the game tree into an equivalent normal-form game. Sequence form assumes that the game is of perfect recall. In perfect recall games, there is a unique path from the root of the game tree to an arbitrary node a. The actions of player k played on such a path are denoted by $\sigma^k(a)$ and are called a **sequence** of choices of player k leading to a. Additionally, let π^k and μ^k denote a pure strategy and a mixed strategy for player k respectively. The **realization weight** of a under μ^k is then the sum of the probabilities $\mu^k(\pi^k)$ over all the pure strategies whose choices match $\sigma^k(a)$ and is denoted by $\mu^k(\sigma^k(a))$. Lastly, let $\beta(a)$ denote the product of chance probabilities on the path to a. We include in this section the sequence-form linear programming definition from Koller et al., 1994 which is also used in its OpenSpiel implementation. The primal problem is as follows:

$$\begin{aligned}
&\underset{\mathbf{y},\,\mathbf{p}}{\text{minimize}} && \mathbf{e}^T\mathbf{p} && (4.1)\\
&\text{subject to:} && -\mathbf{A}\mathbf{y}+\mathbf{E}^T\mathbf{p} \geq \mathbf{0} && (4.2)\\
& && -\mathbf{F}\mathbf{y} = -\mathbf{f} && (4.3)\\
& && \mathbf{y} \geq \mathbf{0} && (4.4)
\end{aligned}$$

Additionally, the dual problem is:

$$\begin{aligned}
\underset{\mathbf{x},\,\mathbf{q}}{\text{maximize}} \quad & -\mathbf{q}^T\mathbf{f} && (4.5)\\
\text{subject to:} \quad & \mathbf{x}^T(-\mathbf{A}) - \mathbf{q}^T\mathbf{F} \leq \mathbf{0} && (4.6)\\
& \mathbf{x}^T\mathbf{E}^T = \mathbf{e}^T && (4.7)\\
& \mathbf{x} \geq \mathbf{0} && (4.8)
\end{aligned}$$

Variables **x** and **y** represent mixed strategies for the individual players. Variable **x** is composed of components that represent $\mu^1(\sigma)$ for each $\sigma \in S^1$, where S^1 is the set of all sequences for player 1. Similarly **y** represents a mixed strategy for player 2 on their sequences. The utility functions of the linear programs determine the expected payoff that player 1 receives. Unconstrained variables **p** and **q** are derived from the dual of the best response linear program which we did not include in this work for the sake of brevity. Let $|U^k|$ denote the number of information sets of player k, and $|D^k|$ denote the number of all choices of player k. Matrices **F**, and **E** are matrices of 1's of dimensions $(1+|U^2|) \times (1+|D^2|)$, and $(1+|U^1|) \times (1+|D^1|)$, respectively. Similarly, vectors **f** and **e** are vectors of 1's of sizes $(1+|U^2|)$ and $(1+|U^1|)$. Constraints (4.3), (4.4), and (4.7), (4.8) ensure that **y** and **x**, respectively, represent mixed strategies. Lastly, matrix **A** of size $(1+|D^1|) \times (1+|D^2|)$ is a payoff matrix. The payoff contribution of sequences σ^1 and σ^2 is determined by those leaves a such that $\sigma^1(a) = \sigma^1$ and $\sigma^2(a) = \sigma^2$. More than one leaf a may define the same pair of sequences due to chance moves. The entries of **A** are then $\sum_{a\in\sigma(a)} \beta(a)h^1(a)$, where $h^1(a)$ is the utility function of player 1 in leaf a. Entries of **A** which do not correspond to any leaves are equal to 0. A detailed explanation of the rationale behind the sequence-form linear programming problem formulation as well as the derivation of the linear programs included in this work can be found in Koller et al., 1994.

4.3 Counterfactual Regret Minimization

Counterfactual regret minimization is an iterative algorithm for finding approximate Nash equilibria in two-player zero-sum extensive-form games. It leverages the fact that in zero-sum games minimizing average overall regret leads to a strategy that converges to a Nash equilibrium. Furthermore, in this algorithm, the minimization of overall regret is achieved through the minimization of immediate counterfactual regret terms which provide an upper bound for the overall regret.

For a more detailed description of the algorithm, we need first to define **counterfactual utility** denoted $u_i(\sigma, I)$ to be the expected utility given that information set I is reached and all players play using strategy profile σ with the exception that player i plays to reach I. Additionally, let $\pi^\sigma(h, h')$ denote the probability of going from history h to history h', and Z the set of terminal nodes, then counterfactual utility is defined as:

$$u_i(\sigma, I) = \frac{\sum_{h \in I, h' \in Z} \pi^{\sigma}_{-i}(h)\pi^{\sigma}(h, h')u_i(h')}{\pi^{\sigma}_{-i}(I)}$$

Then we can define **immediate counterfactual regret** at time T to be:

$$R^T_{i,imm}(I) = \frac{1}{T} \max_{a \in A(i)} \sum_{t=1}^{T} \pi^{\sigma^t}_{-i}(I)(u_i(\sigma^t|_{I \to a}, I) - u_i(\sigma^t, I))$$

where $\sigma^t|_{I \to a}$ is a strategy profile identical to σ except that player i always chooses action a in information set I. Finally, the relationship between **average overall regret** and immediate counterfactual regret is:

$$R^T_i \leq \sum_{I \in \mathcal{I}_i} \max(R^T_{i,imm}(I), 0)$$

The benefit of minimizing immediate counterfactual regrets is that it can be done independently in each information set by determining $\sigma_i(I)$. To this end serves the following formula which defines counterfactual regret for an information state-action pair.

$$R^T_i(I, a) = \frac{1}{T} \sum_{t=1}^{T} \pi^{\sigma^t}_{-i}(I)(u_i(\sigma^t|_{I \to a}, I) - u_i(\sigma^t, I))$$

Each iteration of the algorithm then consists of three stages:

1. Get a strategy for the current iteration which is based on $R^T_i(I, a)$.
2. Compute counterfactual utilities for information sets using the current strategy.
3. Compute $R^{T+1}_i(I, a)$ using counterfactual utilities.

The average strategy obtained from the individual strategies in their respective iterations satisfies the property of minimizing immediate counterfactual regrets, thus minimizing the average overall regret and converging to a Nash equilibrium strategy according to the following theorem:

Theorem In a zero-sum game at time T, if both player's average overall regret is less than ϵ, then $\overline{\sigma}^T$ is a 2ϵ equilibrium.

From the fact that the computation of counterfactual regrets, and counterfactual utilities require full traversal of the game tree, it is easy to see that CFR does not scale as well as algorithms using generalization on information sets. For more information about CFR as well as additional theoretical properties refer to Zinkevich et al., 2007 which served as a source for this section.

4.4 Neural Fictitious Self-Play

The key concept of **general self-play** is the repeated play of two or more agents who gradually improve their strategies over a number of iterations. This is mostly done by finding a best response to the opponent's strategy in each iteration and incorporating this best response strategy into the player's average strategy. In zero-sum games, the average strategy created by such an iterative process converges to a Nash equilibrium strategy.

Fictitious self-play (FSP) is a sample- and machine learning-based class of algorithms that approximate this behavior in extensive-form games. The main motivation behind this approach is solving large domains. In FSP each agent stores two datasets of their experience in self-play: $\mathcal{M}_{RL}$ stores transition tuples $(s_t, a_t, r_{t+1}, s_{t+1})$ for approximation of state-action values, and tuples (s_t, a_t) are stored in $\mathcal{M}_{SL}$ which is used for approximation of agent's own average strategy during the self-play period. An arbitrary reinforcement learning algorithm may be used on $\mathcal{M}_{RL}$ to approximate state-action values. Similarly, any supervised learning algorithm can be used for average strategy approximation. **Neural fictitious self-slay (NFSP)** uses for both reinforcement learning and supervised learning algorithms deep neural networks for their respective purposes.

NFSP introduces a deep neural network $Q(s, a|\theta^Q)$ to predict state-action values using off-policy reinforcement learning. The resulting state-action values define an approximate best response strategy $\beta = \epsilon\text{-greedy}(Q)$ which selects a random action with probability ϵ and otherwise chooses the action that maximizes the predicted state-action value. A separate neural network $\Pi(s, a|\theta^\Pi)$ imitates the agent's own past best response behavior using supervised classification. It maps states to action probabilities and defines the agent's average strategy π. During self-play, the agent uses a mixture of the two strategies which is determined by an anticipatory parameter η in the following way:

$$\sigma = \begin{cases} \epsilon\text{-greedy}(Q) & \text{with probability } \eta \\ \Pi & \text{with probability 1-}\eta \end{cases}$$

This policy assignment is sampled before each playthrough of the game. During evaluation, only the average strategy estimated by network Π is used as it converges to a Nash equilibrium strategy. For a more detailed description of the algorithm refer to Heinrich and Silver, 2016.

Chapter 5
Experimental Evaluation

In our experiments, we model a scenario where the defender has only partial information about the computer network and its vulnerabilities. This is represented by a pair of graphs $G = (N, E_k)$ and $G' = (N, E_k \cup E_h)$. These two graphs model the same network, where G is the game available to the defender without the unknown vulnerabilities and G' is the full game available to the attacker. We also denote the full game G' associated with a partial game G by expression $G' \leftarrow G$. The comparison of the two graphs can be seen in Figure 5.1, where we assume $E_h = \{e_{1,3}, e_{3,5}\}$ (game G is located on the left-hand side of the figure, and G' is located on the right-hand side).

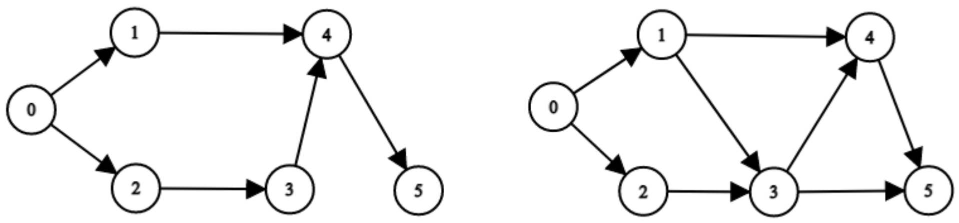

Figure 5.1: Comparison of G and G'

In our experiments, we measure how well an optimal strategy learned by the defender in G translates to game G'. Let $s = (s_1, s_2)$ and $s' = (s'_1, s'_2)$ denote a Nash equilibrium strategy in games G and G', respectively. Additionally, let β'_2 denote a best response strategy to strategy s_1 in game G'. Finally, let $u_1(s_1, s_2|G)$ denote the expected utility for the defender in game G under strategy profile $s = (s_1, s_2)$. Our experiments consist of measuring the defender's expected utility under two strategy profiles $s'_{\text{NE}} = (s_1, s'_2)$ and $s'_\beta = (s_1, \beta'_2)$ in the full game, i.e. $u_1(s'_{\text{NE}}|G')$ and $u_1(s'_\beta|G')$. Strategy profile s'_{NE} models a scenario in which both agents learn to play optimally in their respective games without any prior knowledge of the opponent's strategy. On the other hand, s'_β models a scenario where the defender's strategy is known to the attacker in advance and thus the attacker is able to exploit it. Both scenarios have a basis in real-life. Another property that we observed in some of our experiments is NashConv of strategies defined in Lanctot et al., 2017 as follows:

$$\textsc{NashConv}(\sigma) = \textstyle\sum_i^n \max_{\sigma'_i \in \Sigma_i} u_i(\sigma'_i, \sigma_{-i}) - u_i(\sigma)$$

NashConv is a sum of the increases in utility each player would gain if they played a best response strategy to the rest of the strategy profile instead of their current strategy. It can therefore be interpreted as a distance from a Nash equilibrium because NashConv in a Nash equilibrium is equal to 0. Additionally, if $\textsc{NashConv}(\sigma) = \epsilon$, then σ is at least an ϵ-Nash equilibrium. Consequently, it verifies that the algorithms did converge to an ϵ-Nash equilibrium and that we do not observe diverging strategies. The main limitation of NashConv is that it requires the computation of best response strategies and utilities for strategy profiles which is infeasible for large domains. An approximation of best responses and utilities may be utilized as an alternative in such cases but because NashConv is a relatively fine measurement the value of such results is poor. Additionally, similar information is provided by the comparison of $u_1(s'_{\mathrm{NE}}|G')$, and $u_1(s'_\beta|G')$, therefore NashConv is not included in experiments on large games.

The above-described measurements were obtained on three classes of games:

- trivial games - games solvable by all algorithms, solutions may be validated manually
- medium-sized games - the maximum size of games, where sequence-form linear programming provides a solution
- large games - computationally infeasible for exact algorithms, solved only by reinforcement learning agents

For each of these classes, 10 games were generated by the random generator described in Section 3.3. Hyperparameters for the games were selected based on experiments conducted in Horák et al., 2019. The hyperparameter configuration is described in Appendix B. The evaluation of the experiments is different for large games and for the two smaller-sized classes.

For trivial games and medium-sized games SFLP, CFR, and NFSP were used to compute $s = (s_1, s_2)$ and $s' = (s'_1, s'_2)$. Subsequently, s_1 computed by each algorithm faced off against s'_2 of all of the algorithms. Moreover, $s'_\beta = (s_1, \beta'_2)$ was computed for each of the s_1 strategies. For large games, since SFLP and CFR become infeasible, both s and s' are found using NFSP exclusively, and β'_2 is estimated with the use of deep Q-learning (DQN, Mnih et al., 2015). The expected utilities are approximated by repeated random game tree traversals determined by the given strategy profile. Subsequently, we measure the means of differences in the expected utilities of two strategies to compare their strength. In addition to the means, we also present their confidence intervals. We chose to measure differences because we assumed that the expected utilities may vary across the individual games but the variance of their differences would be low, thus providing a tighter confidence interval. We used 95% confidence intervals. In order to compute confidence intervals, we assumed the differences to be independent samples from a normally distributed population with unknown mean and variance. There is not enough evidence to reject the null hypothesis that the differences come from a normal population. Furthermore, we did measure the mean of NashConv of the computed strategies but we did not determine its confidence intervals as it was not the primary focus of this work and the presented tables would become cluttered.

For NFSP, different configurations of hyperparameters were tested, and the configuration with the best convergence to a Nash equilibrium in game G of trivial and medium size was used for the experiments and is presented in this work. The choice of hyperparameters was primarily inspired by Heinrich and Silver, 2016. Some concessions were made due to time and hardware limitations. Although we have made attempts to choose optimal hyperparameters for our domain, it is possible that hyperparameter configuration with a faster convergence rate exists, as hyperparameter tuning is a complex topic that is outside of the scope of this work.

All results have been obtained on a computer equipped with Intel Core i5-8600K CPU and 16 GB of available RAM. Attempts to increase the performance of NFSP agents by having TensorFlow utilize NVIDIA GeForce GTX 1070 Ti GPU have been made, however, there was about 80% increase in execution times upon utilizing the GPU, therefore the experiments were conducted using solely the CPU.

5.1 Lateral Movement

In this section, we compare the quality of solutions provided by CFR, SFLP, and NFSP. For that purpose, we introduce a baseline utility against a Nash equilibrium opponent and against a best responder. These baseline utilities are determined by strategy profiles $s'_{\text{SFLP}} = (s_1, s'_2)$, and $s^{\beta}_{\text{SFLP}} = (s_1, \beta'_2)$, respectively. Both s_1 and s'_2 in these strategy profiles were found by SFLP. The baseline defender utilities are then denoted $u_1(s'_{\text{SFLP}}|G')$ and $u_1(s^{\beta}_{\text{SFLP}}|G')$. The average difference between the utility of a given strategy profile and the baseline Nash equilibrium utility is presented in our experiments. This average difference is denoted by $\overline{u}_d(s_1, s'_2) = \frac{1}{|\mathcal{G}'|}\sum_{G'\in\mathcal{G}'} u_1(s_1, s'_2|G') - u_1(s'_{\text{SFLP}}|G')$. Similarly, we define the average difference between a given strategy profile and the baseline defender's utility against a best responder to be $\overline{u}^{\beta}_d(s_1, \beta'_2) = \frac{1}{|\mathcal{G}'|}\sum_{G'\in\mathcal{G}'} u_1(s_1, \beta'_2|G') - u_1(s^{\beta}_{\text{SFLP}}|G')$.

Moreover, we examine how well the strategy s_1 translates into game G'. Toward this end, we introduce notation for the average difference between the defender's utility of a strategy s_1 in game G' and the defender's utility of the same strategy in game G to be $\overline{u}_{\Delta G}(s_1) = \frac{1}{|\mathcal{G}|}\sum_{G\in\mathcal{G}} u_1(s_1, s'_2|G' \leftarrow G) - u_1(s_1, s_2|G)$. In $\overline{u}_{\Delta G}(s_1)$, strategy profiles s and s' are always computed by the same algorithm. Finally, we define the same average difference for s_1 and the best response opponent as $\overline{u}^{\beta}_{\Delta G}(s_1) = \frac{1}{|\mathcal{G}|}\sum_{G\in\mathcal{G}} u_1(s_1, \beta'|G' \leftarrow G) - u_1(s_1, s_2|G)$.

These measurements enable us to detect games, where the optimal strategy changes drastically with the addition of hidden vulnerabilities.

		SFLP	CFR	NFSP	NashConv
SFLP	$\overline{u}_d$	0	-0.0055816	-0.0167958	4e-9
	CI	$[0, 0]$	$[-0.0132, 0.002]$	$[-0.0909, 0.0573]$	
CFR	$\overline{u}_d$	-0.0082658	-0.0137724	-0.026565	0.000844
	CI	$[-0.027, 0.0105]$	$[-0.0326, 0.005]$	$[-0.1009, 0.0477]$	
NFSP	$\overline{u}_d$	-0.0136148	-0.0184302	-0.0229209	0.028352
	CI	$[-0.0425, 0.0153]$	$[-0.0479, 0.011]$	$[-0.0991, 0.0532]$	
NashConv		6e-9	0.001772	0.056109	

Table 5.1: $\overline{u}_d(s_1, s'_2)$ in trivial games

In Tables 5.1 and 5.3 $\overline{u}_d(s_1, s'_2)$, its 95% confidence interval (denoted CI) and NashConv of the individual algorithms are presented. The algorithm used to compute s_1 is located in the table row, whereas the algorithm used to compute s'_2 is located in the table column. We also average the obtained differences against all the opponent algorithms to obtain an "opponent independent" $\overline{u}_d(s_1, s'_2)$. These opponent independent $\overline{u}_d(s_1, s'_2)$, as well as $\overline{u}^{\beta}_d(s_1, \beta'_2)$, $\overline{u}_{\Delta G}(s_1)$, $\overline{u}^{\beta}_{\Delta G}(s_1)$ and their 95% confidence intervals are presented in Tables 5.2 and 5.4. Lastly, $\overline{u}_{\Delta G}(s_1)$ and $\overline{u}^{\beta}_{\Delta G}(s_1)$ were approximated on the large games via policy-dependent game tree traversals. These results and their confidence intervals are presented in Table 5.5.

		$\overline{u}_d(s_1, s'_2)$	$\overline{u}^{\beta}_d(s_1, \beta'_2)$	$\overline{u}_{\Delta G}(s_1)$	$\overline{u}^{\beta}_{\Delta G}(s_1)$
SFLP	$\overline{u}$	-0.0075	0	-0.4589	-0.6619
	CI	$[-0.0293, 0.0144]$	$[0, 0]$	$[-0.7821, -0.1357]$	$[-1.0041, -0.3197]$
CFR	$\overline{u}$	-0.0162	-0.4722	-0.3507	-0.6962
	CI	$[-0.0393, 0.0069]$	$[-0.0917, 0.0221]$	$[-0.788, -0.1563]$	$[-1.0012, -0.3912]$
NFSP	$\overline{u}$	-0.0183	-0.0186	-0.4646	-0.6633
	CI	$[-0.0435, 0.0069]$	$[-0.1035, 0.0663]$	$[-0.7648, -0.1645]$	$[-0.9571, -0.3696]$

Table 5.2: Average differences in utilities in trivial size games

		SFLP	CFR	NFSP	NashConv
SFLP	$\overline{u}_d$	0	-0.1789	-0.2149	3.4e-8
	CI	$[0, 0]$	$[-0.4233, 0.0655]$	$[-0.4955, 0.0656]$	
CFR	$\overline{u}_d$	-0.0012	-0.1817	-0.2308	0.000321
	CI	$[-0.0065, 0.004]$	$[-0.4256, 0.0622]$	[-0.5073, 0.0458]	
NFSP	$\overline{u}_d$	-0.0033	-0.1624	-0.2167	0.033666
	CI	$[-0.1402, 0.1336]$	$[-0.4338, 0.1091]$	$[-0.5244, 0.0911]$	
NashConv		9e-9	0.043338	0.000543	

Table 5.3: $\overline{u}_d(s_1, s'_2)$ in medium-sized games

		$\overline{u}_d(s_1, s'_2)$	$\overline{u}^\beta_d(s_1, \beta'_2)$	$\overline{u}_{\Delta G}(s_1)$	$\overline{u}^\beta_{\Delta G}(s_1)$
SFLP	$\overline{u}$	-0.1313	0	-0.1693	-0.6616
	CI	$[-0.2452, -0.0173]$	$[0, 0]$	$[-0.4125, 0.074]$	$[-1.005, -0.3181]$
CFR	$\overline{u}$	-0.1379	-0.0566	-0.3507	-0.7179
	CI	$[-0.2515, -0.0243]$	$[-0.1548, 0.0416]$	$[-0.704, 0.0025]$	$[-1.0425, -0.3933]$
NFSP	$\overline{u}$	-0.1274	-0.0005	-0.3628	-0.639
	CI	$[-0.2579, 0.003]$	$[-0.1812, 0.1801]$	$[-0.6104, -0.1153]$	$[-0.9017, -0.3763]$

Table 5.4: Average differences in utilities in medium-sized games

		$\overline{u}_{\Delta G}(s_1)$	$\overline{u}^\beta_{\Delta G}(s_1)$
NFSP	$\overline{u}_{\Delta G}$	-0.0587	-0.232
	CI	$[-0.215, 0.0975]$	$[-0.5061, 0.0422]$

Table 5.5: Average differences in utilities in large games

To provide concrete values, for our leading example from Figure 3.1, since $u_1(s_1, s_2|G) = 0.1253$ and $u_1(s_1, s'_2|G' \leftarrow G) = -0.2936$, $u_{\Delta G}(s_1) = -0.4189$. Similarly, because $u_1(s_1, s_2|G) = 0.1253$ and $u_1(s_1, \beta'|G' \leftarrow G) = -0.9663$, $u^\beta_{\Delta G}(s_1) = -1.0916$. We can see that the game dynamic has shifted from the defender being able to always prevent the attacker from reaching the terminal node, to being able to do so only sometimes. Additionally, the utility against the best responder is close to being the worst possible. Our experiments provide us with a couple of interesting observations. First, since our assumption that the variance of differences would be very low was not entirely accurate and the number of samples is only 10, the confidence intervals are in some cases quite large, therefore more experiments would need to be conducted in order to achieve high statistical significance. Still, from our observations, there do not seem to be stark differences between the defender's utilities obtained by the individual algorithms used for Nash equilibrium computation. Out of the three algorithms, SFLP provides solutions with the slightest decrease in utility when applied to game G' in medium-sized games but that is presumably due to the fact that s'_2 found by SFLP was weaker than the s'_2 strategies computed by CFR and NFSP not because the s_1 SFLP strategy was stronger. Additionally, when the number of information sets increases, NFSP produces better $\overline{u}_d(s_1, s'_2)$ against the various Nash equilibrium opponents than SFLP and is comparable in terms of $\overline{u}^\beta_d(s_1, \beta'_2)$. This may be due to the fact that NFSP is the only algorithm out of the three which is able to generalize its policy on information sets encountered in G' for the first time, whereas CFR and SFLP play uniform random policies in these information sets. Moreover, CFR provides on average worse solutions than SFLP in both observed classes of games. Second, the $\overline{u}_{\Delta G}(s_1)$ and $\overline{u}^\beta_{\Delta G}(s_1)$ are both quite significant - especially in the smaller classes of games. We believe this is due to the fact that if the graph is relatively sparse, there are **bottlenecks** in the graphs that the defender may completely cover with honeypots. The attacker then has no other choice but to use the honeypotted edges. Whenever edges that enable the attacker to avoid these bottlenecks are introduced in G', the defender's strategy becomes far from optimum and the defender's expected utility decreases significantly. One benefit of this occurrence is that the defender should be able to observe that their policy is performing worse than expected and modify their current policy to adapt to the change in the game dynamics. Additionally, the occurrence of bottlenecks seems to become less common in large games and as a result, the differences in expected utilities of policies obtained on G and G' are smaller but still observable.

In order to obtain better utilities in G', we considered multiple modifications to our approach. First, we considered adding specific edges to G to create G'' which would be closer to G' than G. However, the choice of nodes these edges connect without any prior knowledge of G' is unclear. One such approach would be to consider a complete graph, however, that increases the computational complexity exponentially as the size of the graph increases. Another option is to detect bottlenecks in G and introduce specific edges that enable the attacker to bypass them. This would encourage the defender to adopt a different strategy than exploiting these bottlenecks. A problem with this approach is that in games without bottlenecks, there is no such choice of hidden edges. Fortunately, if that is the case, the $\overline{u}_{\Delta G}(s_1)$ results are better than in graphs with bottlenecks, therefore not altering G might be an option. This direction might be a topic of further research, however, the highlighted problems caused us to pursue a different direction. We chose a modification of G which is simpler and serves the same purpose of encouraging the defender to adopt a more robust strategy. We do this by introducing chance to the game. A honeypot has a fixed probability of a defect. If a honeypot defects, the attacker's advance through the graph is uninterrupted, and the defender receives no information about the attacker's movement. The results of this approach are discussed in Section 5.2. The third avenue is a reinforcement learning approach of online policy adaptation in the full game G'. This approach and its limitations are discussed in Section 5.3.

5.2 Lateral Movement with Defect

The same set of experiments as in the previous section was conducted with the exception that instead of using a Nash equilibrium defender's strategy from game G (i.e. s_1) in game G', a Nash equilibrium strategy from game G'' was used (i.e. s_1''). The difference between G'' and G is that whenever the attacker uses a honeypot, it might defect according to a given probability distribution. When a honeypot defects, the attacker continues in infecting the computer network uninterrupted and the defender is not notified of the attacker's progress. Game G'' does not contain hidden vulnerabilities. Instead of the measurements observed in the previous experiments, we measure the average difference between utilities of s_1'' and s_1 in game G'. We denote this difference against a Nash equilibrium attacker $\Delta\overline{u}_1(s_1'', s_2'|G') = \frac{1}{|\mathcal{G}'|}\sum_{G'\in\mathcal{G}'} u_1(s_1'', s_2'|G') - u_1(s_1, s_2'|G')$. Similarly, the difference in utilities against a best responder is $\Delta\overline{u}_1(s_1'', \beta_2'|G') = \frac{1}{|\mathcal{G}'|}\sum_{G'\in\mathcal{G}'} u_1(s_1'', \beta_2'|G') - u_1(s_1, \beta_2'|G')$. The differences in utilities always relate to strategies learned by the same set of algorithms on both games G and G''. For the experiments presented in this work, we used the defect rate of 10%, however, no major differences were observed when other defect rates were used. It seems that the main difference lies in new information states being reachable, not in the probability of reaching them. In Tables 5.6 and 5.8, are presented $\Delta\overline{u}_1(s_1'', s_2'|G')$ of each of the algorithms against all the others and their 95% confidence intervals. The defender's algorithm is located in the rows of said tables and the attacker's algorithm is in the columns. Tables 5.7, 5.9, and 5.10 depict the differences in utilities averaged across all opponents and their confidence intervals.

		SFLP	CFR	NFSP	NashConv
SFLP	$\Delta\overline{u}_1$	0.1024	0.1036	0.091	1e-8
	CI	$[-0.1283, 0.3331]$	$[-0.1265, 0.3338]$	$[-0.1286, 0.3107]$	
CFR	$\Delta\overline{u}_1$	0.1049	0.1059	0.0955	0.00067
	CI	$[-0.1215, 0.3313]$	$[-0.1199, 0.3318]$	$[-0.1199, 0.3108]$	
NFSP	$\Delta\overline{u}_1$	0.0679	0.0683	0.0568	0.028955
	CI	$[-0.0872, 0.223]$	$[-0.0865, 0.2231]$	$[-0.094, 0.2076]$	
NashConv		7e-9	0.001772	0.049266	

Table 5.6: $\Delta\overline{u}_1(s_1'', s_2'|G')$ in trivial games with defect

		$\Delta\overline{u}_1(s_1'', s_2'|G')$	$\Delta\overline{u}_1(s_1'', \beta_2'|G')$
SFLP	$\Delta\overline{u}_1$	0.099	0.1526
	CI	$[-0.0153, 0.2133]$	$[-0.0945, 0.3997]$
CFR	$\Delta\overline{u}_1$	0.1021	0.1697
	CI	$[-0.01, 0.2142]$	$[-0.0643, 0.4036]$
NFSP	$\Delta\overline{u}_1$	0.0643	0.1028
	CI	$[-0.0131, 0.1417]$	$[-0.0554, 0.2609]$

Table 5.7: Average differences in utilities in trivial size games with defect

		SFLP	CFR	NFSP	NashConv
SFLP	$\Delta\overline{u}_1$	-0.0101	-0.0624	-0.0143	6.8e-8
	CI	$[-0.3552, 0.3351]$	$[-0.4487, 0.324]$	$[-0.293, 0.2643]$	
CFR	$\Delta\overline{u}_1$	-0.033	-0.0554	-0.0048	0.000454
	CI	$[-0.3828, 0.3168]$	$[-0.4434, 0.3326]$	$[-0.2664, 0.2568]$	
NFSP	$\Delta\overline{u}_1$	-0.085	-0.101	-0.0238	0.033948
	CI	$[-0.3174, 0.1473]$	$[-0.3482, 0.1462]$	$[-0.2297, 0.1821]$	
NashConv		3e-9	0.000368	0.04868	

Table 5.8: $\Delta\overline{u}_1(s_1'', s_2'|G')$ in medium-sized games with defect

		$\Delta\overline{u}_1(s_1'', s_2' \mid G')$	$\Delta\overline{u}_1(s_1'', \beta_2' \mid G')$
SFLP	$\Delta\overline{u}_1$	-0.0289	0.2276
	CI	$[-0.2002, 0.1424]$	$[-0.1918, 0.6471]$
CFR	$\Delta\overline{u}_1$	-0.0311	0.2785
	CI	$[-0.2011, 0.139]$	$[-0.0922, 0.6491]$
NFSP	$\Delta\overline{u}_1$	-0.0699	0.1277
	CI	$[-0.186, 0.0461]$	$[-0.0841, 0.3395]$

Table 5.9: Average differences in utilities in medium-sized games with defect

		$\Delta\overline{u}_1(s_1'', s_2' \mid G')$	$\Delta\overline{u}_1(s_1'', \beta_2' \mid G')$
NFSP	$\Delta\overline{u}_1$	0.0106	0.0039
	CI	$[-0.1414, 0.1625]$	$[-0.3131, 0.3209]$

Table 5.10: Average differences in utilities in large games with defect

To provide concrete values, for our leading example, we observed that $u_1(s_1'', s_2' \mid G') = -0.5389$ and $u_1(s_1, s_2' \mid G') = -0.2936$, therefore $\Delta u_1(s_1'', s_2' \mid G') = -0.2453$. Similarly, since $u_1(s_1'', \beta' \mid G') = -0.9663$ and $u_1(s_1, \beta' \mid G') = -0.9663$, $\Delta u_1(s_1'', \beta_2' \mid G') = 0.0$. In this case, the introduction of change provided an equilibrium strategy with lower utility in G' and the utility against the best responder stayed the same. Generally, the defender adopts more robust strategies in G'' than in games without defect, therefore the utility against a best responder improves considerably in trivial and medium-sized games. Interestingly, in medium-sized games, the average utility against a Nash equilibrium opponent decreases, hence this approach does not produce universally superior strategies to the approach presented in Section 5.1. In trivial and large games, however, the utilities against a Nash equilibrium opponent on average improve, although in large games only by a small amount. Similarly, the improvement against a best responder in large games is also lower than in smaller games. This is presumably due to the fact that most information states in game G were reachable and bottlenecks were absent in these large domains, therefore the introduction of defect did not influence the game dynamics in the same way it did in games of smaller sizes. More experiments would need to be conducted to provide tighter confidence intervals and increase the statistical significance of the found results. Nevertheless, according to our findings, s_1'' seems to be more robust than s_1 in G', making it a better choice in most cases.

5.3 Local Policy Search

In addition to improving the expected defender's utility by altering game G, we wanted to also explore an online reinforcement learning approach in the space of all possible policies. That, however, proves to be challenging because of several reasons. First, to begin the exploration of the space of possible policies the agent needs to be able to alter its current policy via actions. In the specification of these actions lies the first major complication. The agent needs to be able to alter the probability of playing an action in each information set by an arbitrary amount. To this end, the policy over all information sets may be represented by a tuple of size $|I| \times |A|$ to which arbitrary tuples of the same dimensions could be added. This makes for a continuous action space. One way to deal with the selection of actions in a continuous action space is to discretize the action space and proceed to solve it using algorithms for discrete problems (Dougherty et al., 1995). The choice of a discretization method is, however, generally unclear and the performance of such an approach may vary dramatically based on the chosen discretization method. Moreover, it severely limits the options available to the agent making it often impossible to learn optimal policies. Another way is to choose the actions via function approximators parametrized by the agent (van Hasselt and Wiering, 2007). This approach faces challenges in the choice of a function approximator (prevalently neural networks) and in the optimization of the function approximator parameters. While both of these approaches are theoretically sound, the size of our problem for nontrivial games makes these approaches most likely infeasible and therefore discourages us from applying them to our domain in this work. The problem is additionally complicated by the fact that without any a priori knowledge of the game, the estimation of a player's utility requires a large number of rollouts in order to achieve reasonable precision. Conversely, if a model of the game was available to the defender, conventional algorithms for finding Nash equilibria might be employed. Consequently, we opted for a localized search around the found solution on the partial game instead of an extensive reinforcement learning approach. This simple approach forsakes optimality guarantees in favor of a feasible size computation and a relatively small number of policy evaluations. The results of the localized search might produce an improvement to the found strategy or confirm that no simple improvement of the policy exists, making it a local maximum.

We assume the attacker's policy to be fixed. We use an iterative algorithm that in each iteration generates a new policy from s'' that differs from s'' only in one information set. This modification takes the form of a 5% increment in a subset of the legal actions and a 5% decrement in a disjoint subset of legal actions of equal size. We generate all combinations of such subsets using combination sum. Subsequently, the new policies are evaluated in game G' against a Nash equilibrium player and a best responder. If an improvement to the defender's utility is obtained, the new improved strategy becomes the optimum strategy and can be modified again in the same fashion. In our experiments, we use exact evaluation. A similar result might be achieved by using an approximate evaluation utilizing repeated game tree traversals, however, there is a minor difficulty arising from the estimate quality. If the increase in utility is small, the approximate evaluation might estimate the utility to be slightly lower than the current maximum and consequently not update the current strategy. Conversely, a minor decrease in utility might be estimated to be a minor increase and the policy might worsen. Since the number of information sets raises rapidly as the size of the game increases, we prioritize information sets with the highest reachability according to the current strategy profile. We do this to explore policies in information sets that contribute to the defender's utility the most. The algorithm terminates after a specified number of iterations or whenever the defender's utility cannot be further improved in the described manner. Additionally, computing combination sum for the large games is infeasible making this version of uninformed search inapplicable to these games. As in the previous section, we present the average difference in utility between the defender's strategy s''_1 and its improved version. We denote the newly found strategy $\hat{s}''_1$. Subsequently, we define $\Delta\overline{u}_1(\hat{s}''_1, s'_2|G') = \frac{1}{|\mathcal{G}'|}\sum_{G'\in\mathcal{G}'} u_1(\hat{s}''_1, s'_2|G') - u_1(s''_1, s'_2|G')$ and $\Delta\overline{u}_1(\hat{s}''_1, \beta'_2|G') = \frac{1}{|\mathcal{G}'|}\sum_{G'\in\mathcal{G}'} u_1(\hat{s}''_1, \beta'_2|G') - u_1(s''_1, \beta'_2|G')$. Additionally, we show 95% confidence intervals of these average utilities, the estimated number of policy evaluations needed during the search, and the percentage of explored information sets of the game. These metrics are presented in Tables 5.11 and 5.12. Due to limited computational resources, we present these differences only on strategies obtained in self-play by CFR and SFLP, i.e. both (s''_1, s''_2) and (s'_1, s'_2) were computed by the same algorithm.

		$\Delta\overline{u}_1(\hat{s}''_1, s'_2\|G')$	$\Delta\overline{u}_1(\hat{s}''_1, \beta'_2\|G')$
SFLP	$\Delta\overline{u}_1$	0.1118	0.6824
	CI	$[-0.0424, 0.2659]$	$[0.3735, 0.9914]$
	Iter.	138	173
	Exp.	100 %	100 %
CFR	$\Delta\overline{u}_1$	0.1226	0.6977
	CI	$[-0.0277, 0.2729]$	$[0.394, 1.0013]$
	Iter.	1210	1479
	Exp.	100%	100 %

Table 5.11: Differences between utilities in trivial games

		$\Delta\overline{u}_1(\hat{s}''_1, s'_2\|G')$	$\Delta\overline{u}_1(\hat{s}''_1, \beta'_2\|G')$
SFLP	$\Delta\overline{u}_1$	0.1934	0.1993
	CI	$[-0.035, 0.4219]$	$[-0.0413, 0.4398]$
	Iter.	15260	17091
	% Exp.	2.5%	1.9%
CFR	$\Delta\overline{u}_1$	0.234	0.2669
	CI	$[-0.0049, 0.473]$	$[0.0197, 0.5141]$
	Iter.	148300	159719
	Exp.	2.5%	2.9%

Table 5.12: Differences between utilities in medium-sized games

For our leading example, we observed that $u_1(\hat{s}''_1, s'_2|G') = 0.02323$ and $u_1(s''_1, s'_2|G') = -0.5389$, therefore $\Delta u_1(\hat{s}''_1, s'_2|G') = 0.5621$. Similarly, since $u_1(\hat{s}''_1, \beta'_2|G') = -0.6589$ and $u_1(s''_1, \beta'|G') = -0.9663$, $\Delta u_1(\hat{s}''_1, \beta'_2|G') = 0.3074$. The observed results show a significant increase in average utilities against both Nash equilibrium opponent and a best responder. In trivial games, all of the information sets are explored, therefore producing a policy with utility close to the utility of the optimal policy in game G'. However, unlike the optimal policy, $\hat{s}''_1$ may be exploitable. Even in medium-sized games, where only a very limited number of information sets are explored, this approach produces major improvements. According to our results, SFLP evaluates fewer policies than CFR presumably due to the fact that its policy assigns positive probabilities to a lesser amount of actions and due to the fact that combinations not representing probabilities are pruned (e.g. combinations with negative values). Nevertheless, the number of policy evaluations is still high. In future work, the relation between the number of policy evaluations needed and expected utility improvement could be explored. If a way to explore the state of possible policies in an efficient way were to be developed and the utility approximation precision was to be increased, this approach would provide policies with the highest utilities.

Chapter 6
Conclusion

Game theory is a powerful framework for reasoning about optimal policies in environments, where multiple agents interact with one another. One of its requirements, however, is having precise information about the studied domain. In network security, this requirement might be difficult to fulfill as not all vulnerabilities present in the computer network may be known to the defender. We model such a scenario in computer security using directed acyclic graphs and game theory. We demonstrate that often the optimal strategy according to the information about the computer network available to the defender yields on average much lower utilities against an attacker with full information than the defender expects. We also compared the strength of the defender's policies obtained by several algorithms in a game with hidden vulnerabilities and found these policies comparable in terms of their strength. Additionally, we have detected that in games, where bottlenecks exist, the expected utility of the defender's policy decreases dramatically. We have introduced a method for the computation of more robust strategies by introducing a chance of honeypot defect into the game and compared the quality of its results with results obtained on games without defect. Even though the consequent policies were in fact more robust and achieved better average utilities against a best responder, they did not always perform better against a Nash equilibrium player preventing it from being a universally superior approach for defender policy computation. We have also highlighted an additional approach for robust policy computation via adding additional edges to the graph available to the defender and showcased the problems of this approach. Furthermore, we have studied an online policy adaptation of the defender's policy against a fixed attacker's policy after deployment into the target network. We have discussed complications related to this method arising from the size of the space of all possible policies and the need for a large number of game playthroughs for accurate expected utility estimates. We have presented an algorithm for an uninformed local policy search that yields on average significantly better policies than those computed on the game without hidden vulnerabilities but is inapplicable to large domains. These findings provide further insight into the problem faced by computer network administrators hardening their designated networks against adversarial attacks and provide tools to improve the quality of a solution obtained with limited information about the vulnerabilities in the computer network. There, however, still exists a number of issues that need to be addressed in future research to increase the scalability of such approaches and decrease the number of playthroughs required for online policy adaptation.

Appendix A

Bibliography

Bhushan, B., & Sahoo, G. (2017). Recent advances in attacks, technical challenges, vulnerabilities and their countermeasures in wireless sensor networks. *Wireless Personal Communications*, *98*(2), 2037–2077. https://doi.org/10.1007/s11277-017-4962-0

Brown, N., Lerer, A., Gross, S., & Sandholm, T. (2019). Deep counterfactual regret minimization.

Christin, D., Mogre, P. S., & Hollick, M. (2010). Survey on wireless sensor network technologies for industrial automation: The security and quality of service perspectives. *Future Internet*, *2*(2), 96–125. https://doi.org/10.3390/fi2020096

Dougherty, J., Kohavi, R., & Sahami, M. (1995). Supervised and unsupervised discretization of continuous features. In *Machine learning proceedings 1995* (pp. 194–202). Elsevier.

Heinrich, J., & Silver, D. (2016). Deep reinforcement learning from self-play in imperfect-information games.

Horák, K., Bošanský, B., Tomášek, P., Kiekintveld, C., & Kamhoua, C. (2019). Optimizing honeypot strategies against dynamic lateral movement using partially observable stochastic games. *Computers & Security*, *87*, 101579.

Hussain, R., & Zeadally, S. (2019). Autonomous cars: Research results, issues, and future challenges. *IEEE Communications Surveys & Tutorials*, *21*(2), 1275–1313. https://doi.org/10.1109/COMST.2018.2869360

Koller, D., Megiddo, N., & von Stengel, B. (1994). Fast algorithms for finding randomized strategies in game trees. *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing*, 750–759. https://doi.org/10.1145/195058.195451

Lanctot, M., Lockhart, E., Lespiau, J.-B., Zambaldi, V., Upadhyay, S., Pérolat, J., Srinivasan, S., Timbers, F., Tuyls, K., Omidshafiei, S., Hennes, D., Morrill, D., Muller, P., Ewalds, T., Faulkner, R., Kramár, J., Vylder, B. D., Saeta, B., Bradbury, J., ... Ryan-Davis, J. (2019). OpenSpiel: A framework for reinforcement learning in games. *CoRR*, *abs/1908.09453*. http://arxiv.org/abs/1908.09453

Lanctot, M., Zambaldi, V., Gruslys, A., Lazaridou, A., Tuyls, K., Perolat, J., Silver, D., & Graepel, T. (2017). A unified game-theoretic approach to multiagent reinforcement learning.

Lockhart, E., Lanctot, M., Pérolat, J., Lespiau, J.-B., Morrill, D., Timbers, F., & Tuyls, K. (2020). Computing approximate equilibria in sequential adversarial games by exploitability descent.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533. https://doi.org/10.1038/nature14236

Nawrocki, M., Wählisch, M., Schmidt, T. C., Keil, C., & Schönfelder, J. (2016). A survey on honeypot software and data analysis. *CoRR*, *abs/1608.06249*. http://arxiv.org/abs/1608.06249

Sayed, M. A., Anwar, A., Kiekintveld, C., Bosansky, B., & Kamhoua, C. (2023). Cyber deception against zero-day attacks: A game theoretic approach. https://doi.org/10.1007/978-3-031-26369-9_3

Shoham, Y., & Leyton-Brown, K. (2008). *Multiagent systems*. Cambridge University Press.

van Hasselt, H., & Wiering, M. A. (2007). Reinforcement learning in continuous action spaces. *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, 272–279. https://doi.org/10.1109/ADPRL.2007.368199

Xin, L., & Bin, D. (2013). The latest status and development trends of military unmanned ground vehicles. *2013 Chinese Automation Congress*, 533–537. https://doi.org/10.1109/CAC.2013.6775792

Zinkevich, M., Johanson, M., Bowling, M., & Piccione, C. (2007). Regret minimization in games with incomplete information. In J. Platt, D. Koller, Y. Singer, & S. Roweis (Eds.), *Advances in neural information processing systems*. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2007/file/08d98638c6fcd194a4b1e6992063e944-Paper.pdf

Appendix B

Experimental settings

In this section we cover hyperparameter settings used in our experiments. The names of the hyperparameters and their values are taken directly from the project's code. For each hyperparameter configuration from Table B.1 a pair of graphs was created, one having enable_hidden parameter set to true (game G' with hidden vulnerabilities), and the other having it set to false (game G without the hidden vulnerabilities). Additionally, whenever defect was introduced to the game, G had the defect_rate hyperparameter set to 0.1. The defect_rate of G' was always set to 0.

Parameter	Trivial	Medium		Large	
num_nodes	5	6	8	16	18
num_honeypots	1	2	3	4	4
num_stages	1	2	1	4	3
density	2	2	2	3	4
num_hidden_edges	1	2	3	5	5

Table B.1: Hyperparameters for graph creation

We used {0, 1, 2, 3, 4, 9, 11, 13, 17, 18} as seeds for the trivial graphs. In the medium-sized graphs 6 nodes case, the set of seeds was {0, 3, 4, 6, 10}, and {15, 20, 37, 42, 60} was the set of seeds in the 8 nodes case. As to the large games, the seeds for the 16 nodes case were {1, 2, 6, 10, 11}, and lastly, for the 18 nodes case, they were {196, 268, 297, 440, 563}. The seeds were chosen in such a way that the sets of edges in the individual graphs were unique.

In terms of the algorithm hyperparameters we used the default OpenSpiel linear programming solver settings for SFLP. Concerning CFR, the number of iterations was 2000 in both trivial games and medium-sized games. Hyperparameters for NFSP are depicted in Table B.2. Amongst the hyperparameters for NFSP are also included hyperparameters for the underlying DQN network for state-action value estimation.

Parameter	Trivial	Medium	Large
num_train_episodes	$1.6 \cdot 10^6$	$2.4 \cdot 10^6$	$1.25 \cdot 10^6$
batch_size	64		
replay_buffer_capacity	$2 \cdot 10^5$		
anticipatory_param	0.2		
hidden_layers_sizes	[64]	[64]	[64, 64, 64]
reservoir_buffer_capacity	$2 \cdot 10^5$		
epsilon_start	0.06		
epsilon_end	0.001		
rl_learning_rate	0.01		
sl_learning_rate	0.01		
min_buffer_size_to_learn	1000		
learn_every	64		
optimizer_str	sgd		
epsilon_decay_duration	$1.6 \cdot 10^6$	$2.4 \cdot 10^6$	$1.25 \cdot 10^6$
loss_str	mse		
update_target_network_every	1000		
discount_factor	1		

Table B.2: NFSP hyperparameters

Regarding policy evaluation in large games, 10^5 random game tree traversals were used in order to approximate the defender's utility. Additionally, Table B.3 shows the hyperparameters used by DQN for the best response policy approximation.

num_train_episodes	$1.28 \cdot 10^5$
hidden_layers_sizes	[64, 64, 64]
replay_buffer_capacity	$2 \cdot 10^5$
batch_size	64
learning_rate	0.1
update_target_network_every	1000
learn_every	10
discount_factor	1
min_buffer_size_to_learn	1000
epsilon_start	1.0
epsilon_end	0.1
epsilon_decay_duration	$1.28 \cdot 10^5$
optimizer_str	sgd
loss_str	mse

Table B.3: DQN hyperparameters for the best response approximation

Appendix C

Code Structure

This section details the code structure of the project. However, for the sake of brevity, we do not cover the entirety of the OpenSpiel source code, only the files newly created for the purposes of this work.

```
openspiel
└─ openspiel
   ├─ games
   │  ├─ lateral_movement.cc
   │  ├─ lateral_movement.h
   │  └─ lateral_movement_test.cc
   └─ python
      ├─ examples
      │  ├─ lm_cfr.py
      │  ├─ lm_deep_cfr.py
      │  ├─ lm_dqn.py
      │  ├─ lm_exploitability_descent.py
      │  ├─ lm_nfsp.py
      │  ├─ lm_policy_evaluation_approximate.py
      │  ├─ lm_policy_evaluation_exact.py
      │  ├─ lm_policy_search.py
      │  └─ lm_sflp.py
      └─ algorithms
         └─ approximate_evaluation.py
```

Files `lateral_movement.cc` and `lateral_movement.h` contain the implementation of the lateral movement domain itself. Tests for the generation of the two information set representations, rewards, as well as tests for integration into the OpenSpiel framework, are located in `lateral_movement_test.cc`. Solvers for Nash equilibrium strategies using the various algorithms mentioned in Chapter 4 can be found in `lm_cfr.py`, `lm_deep_cfr.py`, `lm_exploitability_descent.py`, `lm_nfsp.py` and `lm_sflp.py`. Script `lm_policy_evaluation_exact.py` was used for the experiments conducted on trivial and medium-sized games. Similarly, `lm_policy_evaluation_approximate.py` was used for the experiments conducted on the large domains in combination with the auxiliary programs in `approximate_evaluation.py` and `lm_dqn.py` which were used for the defender's expected utility approximation and best response approximation, respectively. Lastly, `lm_policy_search.py` was utilized for the online improvement of the found policy.