Czech Technical University in Prague

Faculty of Electrical Engineering

Department of Computer Science

Bachelor's Thesis

Modeling Various Defense Actions in Adversarial Anomaly Detection Games

Author: Martin Řepa

Supervised by Ing. Karel Durkota Ph.D.

May 2019

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Řepa**	Jméno: **Martin**	Osobní číslo: **466119**	

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Studijní obor: **Software**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Modelovani Různých Obranných Akci v Hrách pro Adverzativní Detekce

Název bakalářské práce anglicky:

Modeling Various Defense Actions in Adversarial Anomaly Detection Games

Pokyny pro vypracování:

Network security companies often use machine learning techniques to detect IP address with malicious behavior. However, they face problems what to do after the detection happens. They typically block the attacker's IP address for several days. As a consequence, this may lead to visibility loss: the attacker can change his IP address and continue to attack, while we have to put another effort in detecting him. The goal of this thesis is to explore other possibilities than simply blocking the attacker's IP address in a game-theoretic settings.
Guidelines:
1. Study algorithms and principles of malicious behavior detections and game theory.
2. Propose alternative actions for the defender (e.g., drop some replies to the attacker, or increase latency of replies to the attacker, etc.).
3. Propose a game that models the interaction between the attacker and the defender.
4. Design and implement algorithms that solve the proposed game (computes chosen solution concept).
5. Experimentally analyze the algorithms' performance (quality of the solutions, scalability, etc.) on syntetic (and/or the provided datasets).

Seznam doporučené literatury:

[1] Durkota, Karel, et al. "Optimal Strategies for Detecting Data Exfiltration by Internal and External Attackers." International Conference on Decision and Game Theory for Security. Springer, Cham, 2017.
[2] Oliehoek, Frans A., et al. "GANGs: Generative Adversarial Network Games." arXiv preprint arXiv:1712.00679 (2017).
[3] Goodfellow, Ian, et al. "Generative adversarial nets." Advances in neural information processing systems. 2014..

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Karel Durkota, Ph.D., centrum umělé inteligence FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **05.02.2019** Termín odevzdání bakalářské práce: **24.05.2019**

Platnost zadání bakalářské práce: **20.09.2020**

Ing. Karel Durkota, Ph.D.		prof. Ing. Pavel Ripka, CSc.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

.	
Datum převzetí zadání	Podpis studenta

Declaration

I hereby declare that the presented work has been composed solely by myself and that I have listed all sources of information used within it in accordance with the methodical instructions about ethical principles in the preparation of academic theses.

Prague, date *Signature*

Acknowledgement

I want to thank my supervisor Ing. Karel Durkota Ph.D. for numerous and highly constructive consultations, patience and the significant amount of time he devoted to enhancing my thesis. Thanks are also due to all member's of The Stratosphere IPS project for sharing knowledge and creating an inspirational academic background.

Abstract

The aim of this thesis is to propose a game theoretic model for adversarial anomaly detection in cybersecurity environment using increasing latency as primary defender's action which is in most cases tolerable by benign users but lethal for malicious hackers. We also propose and evaluate solving techniques using double oracle algorithm in conjunction with feedforward neural networks modeling defender's actions and show a real application of the model to the DNS data exfiltration problem. The essential contribution originates in providing besides the detection also follow-up instructions in terms of increased latency and thus filling up the gap created by lack of research papers about this topic. We managed to succeed in the proposal of a sufficiently generic model based only on the data set containing samples of benign users.

Keywords: game theory, adversarial games, increasing latency, cybersecurity

Abstrakt

Cílem této práce je navrhnout nový herně teoretický model pro adverziální detekce zaměřený primárně na modelování zvyšování latence jako akce obránce. Zároveň definujeme a hodnotíme postupy určené k vyřešení námi navrženého modelu s pomocí algoritmu double oracle a neuronových sítí použitých k modelování akcí obránce. Největší hodnotu naší práce spatříme ve faktu, že mimo detekce anomálií navrhujeme řešení v reálném čase v podobě zvýšené latence, čímž zároveň vyplňujeme prostor způsobený nedostatkem vědeckých prací na podobné téma. Mimo jiné námi představený model je plně generický a závislý pouze na datech normálního provozu. Na závěr ukážeme aplikaci modelu na reálný problém exfiltrací dat pomocí DNS.

Klíčová slova: teorie her, zvyšování latence, počítačová bezpečnost

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Security companies often offer ways to detect anomalies in a network traffic, but the question about what to do next usually remains unanswered. The real physical action is left to a client itself or companies' administrators doing manual inspections.

One of the answers is for example to block the IP address responsible for the detected anomaly, which is the most common and straightforward approach. The reason why the blocking approach may be suboptimal is that an attacker is often able to change his IP address, which means the whole effort and spent resources for the detection are lost. Also, the detection is seldom 100% certain and in that case, the blocking action might leave the companies' services unavailable for some benign users. Even if the detection is completely certain there might be users standing behind the same Network Address Translation as the attacker does. In that case, the unconditional blocking of the attacker's IP address limits the benign users as well.

Another potential answer is to increase latency for the given IP address, meaning to respond to the packet or forward the packet slower. This seems to be a much better choice since the benign user does not have to necessarily care so much for being a little bit slowed down as long as he receives the request response. On the other hand, in many cases the attacker cares more because usually the attacker needs to send massive amount of packets during the attacks and thus his ability to perform successful attack substantially decreases.

One might argue that most of the current anomaly detection solutions output probability of the connection to be malicious, and therefore this probability might be used as latency value. Unfortunately, this approach would not be entirely correct because the cost of slowing the attacker and the cost of slowing the benign user might vary a lot. For example, there might be a company doing business in the stock market where even milliseconds matter, hence slowing the benign user down does more damage than not slowing the attacker.

Another answer is, for example, redirecting the attacker to honeypots, which are physical or virtual hosts simulating the attacking environment to stall the attacker and learn some information by analyzing his behaviour. This approach would, of course, limit the benign user the same way as the blocking approach would, however this way we could at least learn some useful information about the attacker and the detection effort would not be thrown away. Nonetheless, redirection to honeypots does not seem as very generic solution as honeypots simulating different environment might have different behaviour and also the companies would need to have access to them.

This elaboration makes us consider solely increasing latency as the best and most universal answer. However, this approach has not yet been surprisingly explored. It means further in the thesis we focus primarily on modeling defence strategy with increasing latency. Also, increasing latency includes the blocking approach as well because slowing the attacker thoroughly might be interpreted as blocking the attacker.

Finally, the challenging problem is to compute optimal strategy, which maximizes a trade-off between costs for false positives and increase of network security. For this purpose, we use game theory. We decided to use game-theoretic approach to model an interaction between the attacker and a defender because the game theory is a convenient tool for modeling an environment with adverse actors trying to achieve contradictory goals such as cyber security. Also, using game theory, we manage to compute the optimal strategy even against an adaptive attacker. The reason is that game theory assumes both players to be completely rational and the rational attacker would try to shape the attacks in response to the defender's algorithm.

Generally, using game theory, we are able to compute optimal strategy such as no actor is able to get more gain by deviating to any other action. Although to accomplish such result we need to model the attacker and the defender correctly so the environment represents the real world. Usually, the players have finite sets of actions. However, there are many ways how to increase the latency, and thus the defender has an infinite set of actions. That is why we are actually dealing with quite arduous task.

We accomplish to propose a generic solution and that is why the number of application examples is almost inexhaustible. Basically, the application could take place in any adversarial environment within real-time internet communication. Example use cases might be represented by scanning network machines' ports, man in the middle attacks or by data exfiltration. We also plan to show the example application of our proposed model, which is the use case of data exfiltration using the DNS requests, where we show how our model stands using the real world data.

1.2 Goals

The goal of this thesis is to propose a game-theoretic model where we consider blocking and slowing down (increasing latency) the attacker as defender's actions. We focus on modeling general solution which could be used in many various situations based only on provided data set.

The second part of the thesis is dedicated to designing and implementing algorithms that compute the optimal solution to the proposed model. Lastly, we experimentally analyze the algorithms' performance using synthetic data as well as real data representing DNS traffic within the CTU network.

1.3 Thesis Structure

Initially, in Chapter 2, we analyze related work. Furthermore, we describe terms used in the thesis and their definitions in Chapter 3, so the reader has all needed background to comprehend content at its full extent. We describe the notation and fundamentals of game theory in Section 3.1; then we outline the idea behind Linear programming in Section 3.2. Lastly, we bring to mind the basics of feedforward neural networks in Section 3.3.

Further, we introduce our game-theoretic models and propose solving algorithms in Chapter 4. Firstly we propose a game which models increasing latency as the defender's action in Section 4.1. After that, in Section 4.2, we define a second game which deals only with blocking approach and discuss differences.

In Chapter 5, we experimentally analyze the performance of proposed algorithms. We study the influence of quality of best responses on double oracle algorithm convergence in Section 5.2. In Section 5.3, the optimality and scalability of our solving techniques are evaluated using synthetic data set. In Section 5.4 a case study about the DNS data exfiltration is presented using real traffic data from the CTU network.

Finally, we conclude the thesis in Chapter 6 and describe an implementation overview in Appendix A.

Chapter 2

Related Work

Surprisingly, reviewing the past literature, researchers have not addressed the problem mentioned in the motivation regarding the latency. A key problem with much of the related literature is that the vast majority of papers focus on either attacker or non-attacker classification. As already mentioned in the introduction this approach can hardly be transformed to slowing the attacker down by increasing latency.

A tiny similarity might be found in a rate limiting approach. The rate limiting is used to control the amount of incoming and outgoing traffic to or from a network given some configuration thresholds like for example a maximum number of allowed request in a certain time slot[12, 10]. However, rate limiting shares the same problem as the blocking action as it is fundamentally the same thing after the user exceeds the threshold.

Nonetheless, usage of game theory to model the environment between the attacker and the defender has been shown in [16]. L. Dritsoula et al. offer a game-theoretic approach for the detection of the attackers so the adaptive attackers cannot shape their attacks in response to the defence. However, this case also does not consider increasing latency.

Overall the significant lack of research papers about slowing the attacker's traffic down is one of our motivations to produce this work.

Chapter 3

Background

3.1 Game Theory

3.1.1 Basic Terminology

We use a notation from Multiagent Systems, a book by Shoham et al.[13].

Definition 3.1.1 (Normal Form Game)**.** Two-player normal form game G is a triplet $G = \langle N, A, u \rangle$ where

- $N = \{1, 2\}$ is a set of 2 players (also called actors) indexed by i.
- $A = A_1 \times A_2$ is an action profile. A_i represents set of all available actions for player i.
- $u = (u_1, u_2)$ is a profile of utility functions. Utility function for player i is $u_i : A \to \mathbb{R}$

Example of the two-player game can be seen visualized as a game matrix in Table 3.1, where player 1 chooses one of two row actions, and player 2 chooses one of two column actions. Final payoffs are specified inside each cell.

Definition 3.1.2 (Strategies)**.** Strategy profile S is denoted as $S = S_1 \times S_2$, where S_i is a set of all probability distributions over actions A_i. Strategy $s_i \in S_i$ is called *pure* if positive probability is assigned only to

	player 2	
player 1	1,2	4,0
	1,2	4,0

Table 3.1: An example of normal form game

one action. If more than one action is played with positive probability we say that the strategy is *mixed*.

We define s_{-i} as a strategy of player i's opponent. It means $s_{-1} = s_2$ and $s_{-2} = s_1$ given strategy profile $s \in S$.

Definition 3.1.3 (Expected Utility Function)**.** In Equation 3.1, we override utility functions $u_i : S \to \mathbb{R}$ to express player i's expected utility.

$$u_i(s) = \sum_{a \in A} u_i(a) \prod_{j=1}^{2} s_j(a_j) \tag{3.1}$$

where $s_i(a_j)$ is a probability of player i to play action a_j given strategy profile $s \in S$.

Definition 3.1.4 (Best Response)**.** Given strategy profile $s \in S$ we say that a best response of player i is strategy $s_i \in S_i$ such as there is no other strategy $s_i^{'} \in S_i$ for which following Expression (3.2) would be true. The set of all best responses for given strategy profile $s \in S$ and player i is denoted as $BR(s_{-i})$.

$$u_i(s_i^{'}, s_{-i}) > u_i(s_i, s_{-i}) \tag{3.2}$$

Definition 3.1.5 (ε-Equilibrium)**.** Given parameter $\varepsilon \in \mathbb{R}^+$, strategy profile $s \in S$ is said to be ε-equilibrium if nobody is able to get more gain than ε by deviating. Formally described in Equation 3.3. If $\varepsilon = 0$ then s is called Nash equilibrium. We denote NE(G) as a set of Nash equilibria given normal form game G.

$$u_i(s) \geq u_i(s_{-i}, s_i) - \varepsilon \qquad \forall i \in N, \quad \forall s_i \in BR(s_{-i}) \tag{3.3}$$

Theorem 3.1.1 (Nash's Existence Theorem)**.** *It can be proven that for each game with a finite number of players and a finite action profile at least one Nash equilibrium must exist[2].*

Definition 3.1.6 (Strategic Equivalence)**.** We call two normal form two-player games strategically equivalent if they possess exactly the same set of Nash equilibria.

Claim 1. *Given real and non-negative parameter k, two normal form two players games $G^{'}$ and $G^{''}$ for which Equation* (3.4) *is true are strategically equivalent.*

$$u_i^{'}(s) = k \cdot u_i^{''}(s) \qquad \forall i \in N, \forall s \in S \tag{3.4}$$

Multiplication by positive constant k may represent simply a change of units in the whole game matrix.

Claim 2. *Having normal for game $\Psi = \{\{1,2\},\{S_1,S_2\},\{u_1,u_2\}\}$ if we pair random constants c_j to pure strategies $s_j \in S_2$ and define $\forall s_i \in S_1 \quad u_1^{'}(s_i,s_j) = u_1(s_i,s_j) + c_j$ then normal form game $\Psi^{'} = \{\{1,2\},\{S_1,S_2\},\{u_1^{'},u_2\}\}$ and original game Ψ are strategically equivalent.[7]*

Definition 3.1.7 (Maxmin and Minmax Strategies)**.** Given strategy profile $s \in S$, a maxmin strategy for player i is $\arg\max_{s_i}\min_{s_{-i}} u_i(s_i, s_{-i})$ and a maxmin value for player i is $\max_{s_i}\min_{s_{-i}} u_i(s_i, s_{-i})$.

A minmax strategy for player i **against** player $-i$ is computed as $\arg\min_{s_i}\max_{s_{-i}} u_{-i}(s_i, s_{-i})$ and alternatively a minmax value for player $-i$ is $\min_{s_i}\max_{s_{-i}} u_{-i}(s_i, s_{-i})$.

Definition 3.1.8 (Zero Sum Games)**.** Finally, we will refer to zero sum games which we describe as two-player normal form games where Equations (3.5) are true.

$$u_1(s) + u_2(s) = 0 \quad \forall s \in S \tag{3.5}$$

Fact 1 (Minimax Theorem)**.** *Computing Nash Equlibrium of finite zero sum two-player game is equivalent to finding the maxmin and corresponding minmax strategies[1].*

3.1.2 Double Oracle

Even though finite zero-sum games might be solved using linear programming in polynomial time, the same approach cannot be applied for games with enormously big sets of actions due to performance issues. That is why we also mention an algorithm called double oracle[8], which converges to ε-equilibrium. Double oracle is used for games where the game matrix cannot be effortlessly constructed.

The general overview of double oracle algorithm applied to two-player zero sum game can be seen in Algorithm 1. In Algorithm, G^j represents game in iteration j and A^j denotes action profile of game G^j. The algorithm starts with 1×1 game matrix, which is being expanded every iteration. Every iteration the game (considering only actions in the current game matrix) is solved using any existing techniques (e.g. linear programming), providing strategy profile s which is used to find best responses within the original game. If it is not possible for any player to gain more utility than *epsilon* by deviating to the best responses, the algorithm terminates. Otherwise the best responses are added to the game matrix and next iteration starts.

Algorithm 1 Double Oracle

```
$G^1 \leftarrow$ random subgame of $G$ with $1 \times 1$ game matrix
$j \leftarrow 1$
$s \leftarrow \mathrm{NE}(G^j)$
**while** $(\exists i \in N : u_i(s_{-i}, BR^G(s_{-i})) > u_i(s) + \varepsilon)$ **do**
    $A_i^{j+1} \leftarrow A_i^j \cup \{BR^G(s_{-i})\} \qquad \forall i \in N$
    $j \leftarrow j + 1$
    $s \leftarrow \mathrm{NE}(G^j)$
**end while**
**return** $s$
```

3.2 Linear Programming

A linear program (LP) is defined by a set of real valued variables, a linear objective function and a set of linear constraints. More formally LP can be written in a matrix form e.g. as written in Equations (3.6):

$$\begin{aligned} \text{maximize} \quad & \mathbf{w}^T\mathbf{x} \\ \text{subject to} \quad & \mathbf{Ax} \leq \mathbf{b} \\ & \mathbf{x} \quad \geq \mathbf{0} \end{aligned} \tag{3.6}$$

where $x \in \mathbb{R}^n$ is a variable vector, $w \in \mathbb{R}^n$ is a vector containing the weights, $A \in \mathbb{R}^{m \times n}$ is a matrix of constants and $b \in \mathbb{R}^m$ denotes a vector of constants as well.

Finally every linear program (also called primal problem) has a corresponding dual problem which shares the optimal solution. The dual problem of above primal problem is shown in Equations (3.7):

$$\begin{aligned} \text{minimize} \quad & \mathbf{b}^T\mathbf{y} \\ \text{subject to} \quad & \mathbf{A}^T\mathbf{y} \geq \mathbf{w} \\ & \mathbf{y} \quad \geq \mathbf{0} \end{aligned} \tag{3.7}$$

It can be proven that every linear program can be solved in worst-case polynomial time[3]. It is useful to realize that all constraints are linear and thus creating a hyperplanes in a space $\mathbb{R}^n$. Since the objective function is also linear the feasible region is a convex polytope and a local optimum is always a global optimum[9].

3.3 Artificial Neural Networks

Artificial Neural Networks (ANN) are computing systems markedly inspired by biological neural networks. One of the most significant contribution of ANN originates from Universal Approximation Theorem[4], which says that every continuous function can be approximated using ANN.

All ANN consist of arbitrary number of layers with also arbitrary number of neurons in each layer. The most basic ones are called feedfor-

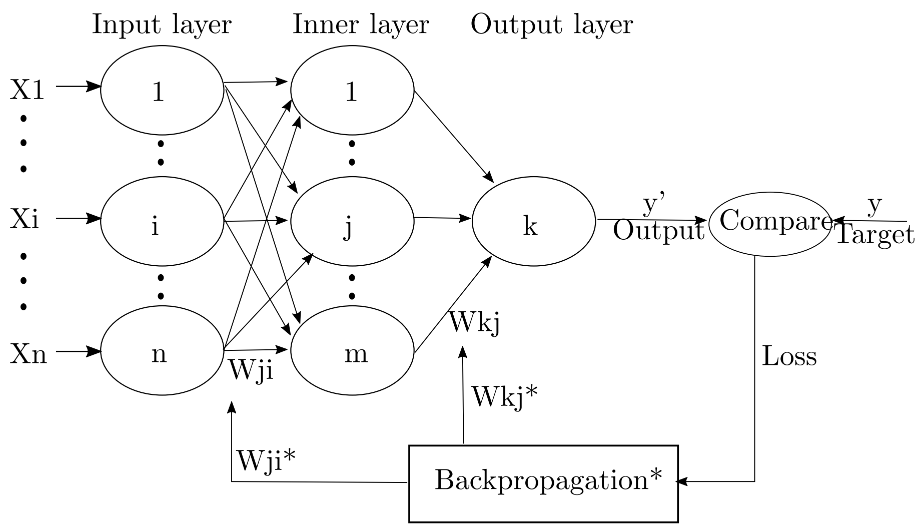


Figure 3.1: Backpropagation process of neural networks

wards taking as input a vector of n features, which are being forwarded through the neural network and returning an output vector of l features. Each neuron is adding a weight and each layer adds a bias on top of it. The idea is to adjust all weights and biases by back-propagating of gradient descent of custom loss function measuring quality of the output features given labeled data set. Described process is outlined in Figure 3.1.

Chapter 4

Game Models

Chapter structure Initially, we introduce a final model which deals with increasing latency as defender's action in Section 4.1. Also, we propose solving algorithms to compute ε-equilibrium in Subsection 4.1.4 and discuss assumptions and limitations in Subsections 4.1.5 and 4.1.6. Lastly, in Section 4.2, we propose a model taking into account only the blocking approach to discuss the differences in Subsection 4.2.3.

4.1 Latency Model

4.1.1 Model Definition

Definition 4.1.1. We define an altered normal form game $\mathcal{G}$ as a tuple $\mathcal{G} = \langle N, A, u, n, D, \alpha_b, \alpha_m, C_b, C_m, P_m, P_b \rangle$ where:

- $N = \{ a, d \}$ is a set of two 2 players (actors) - an attacker a and a defender d
- $n \in \mathbb{N}$ is a dimension of space from which the attacker chooses an attack. In other words each attacker's action is $a \in \mathbb{R}^n$
- $A = A_a \times A_d$ is an action profile where A_i is an action set for player i. An exact annotation can be observed in Equations (4.1).

Player's actions are further described in Subsection 4.1.3.

$$A_a = [0,1]^n \tag{4.1a}$$

$$A_d = \{\, l \,|\, l : A_a \to [0,1] \,\} \tag{4.1b}$$

- $D = \{\, d \,|\, d \in [0,1]^n \,\}$ is provided data set with k number of features from $\mathbb{R}^n$ representing only benign users.
- In Equation 4.2, we define function $h_m : [0,1] \to [0,1]$, which says how certain latency affects the attacker's gain. Function h_m takes latency $l \in [0,1]$ as an input and outputs a number which describes a severity of this latency. Outputting zero denotes that the attacker gets no gain if the defender assigns the latency l to the attacker. Outputting one represents no effect of the latency to the attacker's gain.

  $$h_m(l) = (1-l)^{\alpha_m} \tag{4.2}$$

  By various settings of constant $\alpha_m \in \mathbb{R}^+$ we can change the effect of latency l to the attacker's gain. Visualized in Figure 4.1a.
- Similarly we define function $h_b : [0,1] \to [0,1]$ in Equation 4.3, which says how cost for false positives increases by increasing latency for benign users. The function h_b again takes latency l as an input and outputs a severity of this latency for benign users. Output one is interpreted as the most severe latency. On the other hand output zero represents no effect of the latency l at all.

  $$h_b(l) = l^{\alpha_b} \tag{4.3}$$

  Again, by various settings of constant $\alpha_b \in \mathbb{R}^+$ we can change the effect of the latency l to the cost for slowing benign users down with latency l. Visualized in Figure 4.1b.
- $C_b \in \mathbb{R}^+$ stands for cost for blocking all benign requests. Usually this number represents cost for denying all requests.
- $C_m \in \mathbb{R}^+$ stands for paid cost when the attacker manages to perform not detected attack in its full extent. Usually this number expresses real value of the company we try to defend.

- $P_m \in [0,1]$ stands for a portion of malicious requests found in a traffic collected over some time T. Following expression $P_m+P_b=1$ must be true.
- $P_b \in [0,1]$ stands for a portion of benign requests found in a traffic collected over the same time T. Following expression $P_m + P_b = 1$ must be true.
- $u = u_a \times u_d$ is an utility profile consisting of payoff functions $u_i : A \to \mathbb{R}$ for player i. Each player wants to maximize his payoff function. Definitions of our utility functions can be seen in Equations (4.4).

$$u_a(f,l) = + R(f) \cdot h_m(l(f)) \cdot P_m \cdot C_m \tag{4.4a}$$

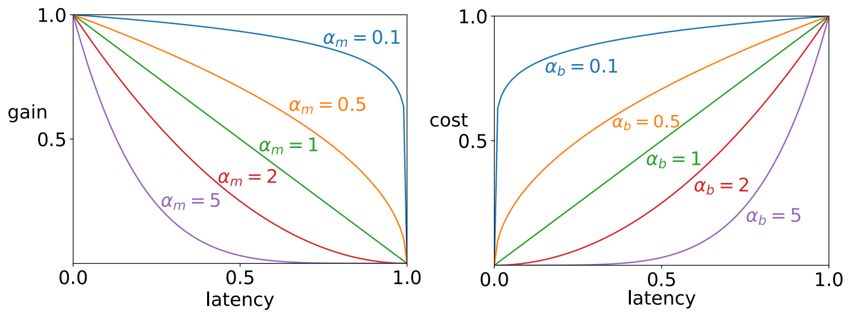
$$\begin{aligned} u_d(f,l) = &- [R(f) \cdot h_m(l(f)) \cdot P_m \cdot C_m] \\ &- [(\sum_{d \in D} h_b(l(f)) \cdot p(d)) \cdot P_b \cdot C_b] \end{aligned} \tag{4.4b}$$

$$R(f) = \prod_{j=1}^{n} f_j \tag{4.4c}$$

where function $p : D \to [0,1]$ represents probability distribution over features in provided benign data set D. In Equation 4.4c, function $R : A_a \to \mathbb{R}$ denotes the attacker's reward for playing his action. See detailed explanation of utility functions in Subsection 4.1.2.

4.1.2 Utility Functions

We see in Equation (4.4a), that the attacker wants to maximize all his features since part of his payoff function is a product of his action features. However, the attacker still tries not to be detected - minimize gotten latency - in order to get maximum payoff. This setup seems to be reflecting the real world in case we choose proper features. Finally, the attacker's payoff is also multiplied by portion of malicious requests in real world P_m as well as constant C_m to take into consideration count of real malicious requests and the potential gain the attacker might obtain.



(a) Constant α_m describing the attacker's gain lost based on increasing latency

(b) Constant α_b describing cost of increasing latency for benign requests

Figure 4.1: The effect of several values of constants α_m and α_b.

Secondly, in Equation (4.4b), we might observe that the utility of the defender consists of negative value of the attacker's utility to "punish" the defender for not detecting the attacker. Second part of the defender's utility is a *false positive part* which is increasing by marking benign data as malicious. As the result the defender tries to find a compromise in slowing the attacker and benign users.

The effect of increasing latency for either malicious or benign requests is various for various values of constants α_m and α_b. Formally this definition can be seen in Equations (4.1a) and (4.1b). This variation is also shown visually in Figure 4.1. We chose parabolas to model this relationships, because we find parabolas to be finely general. What it means is that special cases such as when the attacker needs to be notably slowed or false positives need to be reduced to zero are easily modeled by setting these constants appropriately. For example, if we think that the attacker does not mind being slowed for the reason that he still carries out successful attack, we set constant α_m closed to zero. On the other hand by setting constant α_b close to zero we say that benign users must not be prominently slowed even at the cost of letting the attacker perform rewarding attack.

4.1.3 Players' Actions

We see in Equations (4.1), that the attacker chooses a feature point which is represented by set of features $\in \mathbb{R}^n$ between 0 and 1. The attacker must try to choose the features as highest as possible, because his reward is defined with the product of these features in Equation (4.4c).

On the other hand, the defender chooses a function which assigns latency $l \in [0, 1]$ to each feature point. The output $l = 0.0$ might be interpreted as doing nothing (not increasing latency at all) and $l = 1.0$ as blocking given input features.

4.1.4 Solving the Game

Solving the game in the form as it was defined is not a trivial task mainly because of two major reasons. The first is that we are dealing with a general sum game for which the problem of computing Nash equilibrium is **PPAD**-complete[11]. The second concern originates from the fact of having action sets with infinite dimensions, which means that the entire game matrix cannot be constructed. Due to these problems we need to transform the original game to a zero sum game to ease computing of the Nash equilibrium.

Definition 4.1.2 (Transformed Zero Sum Game $\mathcal{G}'$)**.** We denote a transformed zero sum game $\mathcal{G}'$ as almost identical to $\mathcal{G}$ except that $\mathcal{G}'_u$ is slightly altered and can be seen in Equations (4.5).

$$u_a(f, l) = +R(f) \cdot h_m(l(f)) + (\sum_{d \in D} h_b(l(d)) \cdot p(d)) \cdot C \tag{4.5a}$$

$$u_d(f, l) = -R(f) \cdot h_m(l(f)) - (\sum_{d \in D} h_b(l(d)) \cdot p(d)) \cdot C \tag{4.5b}$$

$$C = \frac{P_b \cdot C_b}{P_m \cdot C_m} \tag{4.5c}$$

Theorem 4.1.1. *Original game $\mathcal{G}$ and transformed game $\mathcal{G}'$ are strategically equivalent.*

Proof. We prove strategic equivalence of games $\mathcal{G}$ and $\mathcal{G}'$ in three steps.

(i) Firstly, in Equation (4.6), we define constant c_l for each pure strategy $l \in S_2$. Then by applying Claim 2 to original game $\mathcal{G}$ we get transformed strategically equivalent game Γ with utility functions defined in Equations (4.7). In other words we added the *false positive part* also to the attacker's utility.

$$c_l = (\sum_{d \in D} h_b(l(d)) \cdot p(d)) \cdot P_b \cdot C_b \tag{4.6}$$

$$\begin{aligned} u_a(f,l) = &+ R(f) \cdot h_m(l(f)) \cdot P_m \cdot C_m \\ &+ [(\sum_{d \in D} h_b(l(d)) \cdot p(d)) \cdot P_b \cdot C_b] \qquad (4.7a) \\ u_d(f,l) = &- [R(f) \cdot h_m(l(f)) \cdot P_m \cdot C_m] \\ &- [(\sum_{d \in D} h_b(l(d)) \cdot p(d)) \cdot P_b \cdot C_b] \qquad (4.7b) \end{aligned}$$

(ii) Secondly, according to Claim 1, we multiply both utility functions in Equations (4.7) with $\frac{1}{P_m \cdot C_m} \in \mathbb{R}^+$ and receive strategically equivalent game $\mathcal{G}^{'}$ with utility functions defined in Equations (4.8)

$$u_a(f,l) = +R(f) \cdot h_m(l(f)) + (\sum_{d \in D} h_b(l(d)) \cdot p(d)) \cdot C \tag{4.8a}$$

$$u_d(f,l) = -R(f) \cdot h_m(l(f)) - (\sum_{d \in D} h_b(l(d)) \cdot p(d)) \cdot C \tag{4.8b}$$

$$C = \frac{P_b \cdot C_b}{P_m \cdot C_m} \tag{4.8c}$$

(iii) Finally since original game $\mathcal{G}$ is strategically equivalent to game Γ and Γ is strategically equivalent to $\mathcal{G}^{'}$ we are allowed to say that original game $\mathcal{G}$ and final game $\mathcal{G}^{'}$ are also strategically equivalent.

$\square$

We managed to get strategically equivalent zero sum game $\mathcal{G}^{'}$ and that is why we might solve this game and use the outcoming Nash equilibria strategies in original game $\mathcal{G}$. We successfully moved from **PPAD**-complete problem to **P**-class problem.

Linear programming may be used to solve finite zero-sum games by solving a minmax strategy as stated in background Chapter 3, nonetheless the dimension of strategy profile $A \in \mathcal{G}'$ is not finite, thus the only way to use LP is to discretize both players' action sets for better performance in exchange for optimality loss. The question then arises as how much we need to discretize in order to keep good trade-off between the performance and the optimality.

For example, we can see a definition of the minmax strategy for discretized game with $n = 2$ in Equation (4.9a) where we discretize each axis in R^2 space from zero to one by one-hundreths points. In this case we assign to the attacker gain equalled to exact product of his discretized features. Even though, for maximum correctness the region around discretized features should be taken into account as well. However, we aim to use this solving technique primarily for optimality comparison with double oracle approach and that is why we find this definition satisfactory.

$$\begin{aligned}\min_{\theta \in \Theta} \max_{s_a^T \in S_a^T} \sum_{x \in A_a^T} \sum_{l \in L} [\theta(x,l) \cdot s_a^T(x) \cdot h_m(l) \cdot R(x)] + \\ + (\sum_{d \in D} \sum_{l \in L} \theta(d,l) \cdot p(d) \cdot h_b(l)) \cdot C \qquad (4.9a)\end{aligned}$$

$$\begin{aligned}\mathbb{X} &= \{\, 0, 0.01, 0.02, \ldots, 0.98, 0.99, 1 \,\} \\ A_a^T &= \mathbb{X}^2 \text{ is a set of discretized attacker's actions} \\ S_a^T &\text{ is set of all probability distributions over actions } A_a^T \qquad (4.9b)\end{aligned}$$

$$\begin{aligned}L &= \{\, 0, 0.25, 0.5, 0.75, 1 \,\} \text{ is set of discretized latencies} \\ \Theta &= \{\, \theta \mid \theta(x,l) \in \mathbb{R}^+ \text{ determines probability of playing} \\ &\quad \text{latency } l \in L \text{ against features } x \in A_a^T; \sum_{l \in L} \theta(x,l) = 1 \,\} \qquad (4.9c)\end{aligned}$$

Finally the precise minmax problem in Equation (4.9a) might be solved in polynomial time by solving linear program defined in Equation (4.10).

$$\begin{aligned} \min \quad & V \\ \text{s.t.} \quad & \sum_{l \in L} \theta(x,l) \cdot h_m(l) \cdot R(x) + \\ & \quad + \sum_{d \in D} \sum_{l \in L} \theta(d,l) \cdot p(d) \cdot h_b(l) \cdot C \leq V \quad && \forall x \in A_a^T \\ & \sum_{l \in L} \theta(x,l) = 1 && \forall x \in A_a^T \\ & \forall l \in \ L \quad \theta(x,l) > 0 && \forall x \in A_a^T \end{aligned} \tag{4.10}$$

Double oracle algorithm is a better approach to solve the game $\mathcal{G}'$, because double oracle is able to compute $\varepsilon - equilibrium$ even in games with infinite action sets. Nevertheless, at the same time there are some challenges which need to be encountered. We need to figure out how to find attacker's and defender's best responses.

To find the defender's best responses we need to first find out how to even represent defender's actions. So far we know that it should be a function $l : A_a \rightarrow [0,1]$. There are plenty of options such as Support Vector Machine (SVM)[6] or clustering using n-dimensional ellipsoidal prototypes[5] or others [14, 15]. Nevertheless, we decided to go with artificial neural networks mainly because of the possibility to model every continuous function as we mentioned in background Section 3.3. This means every defender's action $l \in A_d$ is actually a neural network which takes n features as an input and outputs a predicted latency.

Even though in theory there is a viability to approximate every continuous function using neural network, many challenges rise up during the effort to do so. Those are mainly implementation details (see implementation overview in Appendix A) but they substantially affect the final performance since double oracle algorithm expects discovery of true best response in every iteration of the algorithm. That is why one of our experiments (Experiment 5.2) focuses on analyzing how quality of best responses influences convergence of double oracle algorithm.

A labeled data set containing all used classes is usually needed for supervised learning of neural networks, however when we train our neural networks we are actually searching for a best response given some strategy profile $s \in S$ describing the attacker's features. Thus, we might use these features as malicious training data in each iteration of double oracle. Finally, to really converge to best responses we use negative defender's utility function as neural networks' loss function. This way by minimizing the loss function we maximize defender's utility function.

As an unbiased evaluation of training models there are often used validation data sets to avoid overfitting (i.e. learning training features extremely good but fail every other classification). However, in our case we are searching for a best response given precise attacker's features, that is why we do not actually need validating data set to train our neural networks. Another reason is that we assume our benign data set to contain sufficient number of records to fully represent benign users (see assumptions in Subsection 4.1.5).

The last complication originates from searching for the attacker's best response. One way is to again discretize the attacker's action set and then go through all the actions and simply choose the best one. This approach is formalized in Equation (4.11) using attacker's discretized actions from Equations (4.9b). Expression $s_d(x)$ represents a probability of defender playing action x given strategy profile $s \in S$.

$$\max_{a_a \in A_a^T} \sum_{x \in A_d} u_a(a_a, x) \cdot s_d(x) \tag{4.11}$$

Nevertheless, discretizing the attacker's action set is far away from ideality as traversing the whole space of actions $\mathbb{X}^n$ and choosing the best one is computationally unrealistic for $n > 2$. One approach would be to discretize the attacker's actions more sparsly in order to compute the best response for $n > 2$. Although, we assume that in that case the attacker's best response would not be the best one.

Alternatively we might utilize a similar idea neural networks use and that is gradient descent. Each time we search for the attacker's best response we create initial vector of k random actions, calculate their utilities given defender strategy profile s_d and update those actions using gradient descent to approach the attacker's maximum payoff. Finally, after a certain number of updates, we choose an action providing the best utility as the best response.

Finally we introduce variables ε_a, $\varepsilon_d \in \mathbb{R}^+$ which we use in the termination condition of double oracle. The variable ε_a states how much the attacker's best response must be better than the value of Nash equilibrium to continue to the next iteration of the algorithm. Similarly the variable ε_d determines how much better the defender's best response must be.

4.1.5 Assumptions

Our model assumes both players to be completely rational and possessing entire game information when making decisions. There is also an assumption for provided data set to be the purest as possible, meaning without any malicious requests and with a sufficient number of records to represent all benign users. The fact of using data set containing only benign requests might be considered an advantage of our model for specific use-cases, because we realize that getting labeled data set often complicates the situation.

4.1.6 Limitations

On the other hand, we as well recognize limitations of our model. Firstly, the model gets stochastic in its goal when setting both constants α_m and α_b contradictorily. That is for example in situation when specifying that the attacker receives extremely huge gain when he is not substantially restricted and at the same time the defender pays non trivial cost for any restriction of benign users. This setup is for exmaple represented by setting both constants α_m and α_b close to zero.

Secondly, one must be careful when trying to set already mentioned constants α_b and α_m to the edge of its range. The reason is that in order to train defender's neural networks properly we need to use defender's utility function as loss function we are minimizing in training phase. That way we might witness exploding gradients during the backpropagation because the derivative of the loss function gets extremely big. Fortunately this issue is solvable by setting and changing a learning rate in the training phase appropriately, however the attention must be paid.

Lastly, somebody could see the limitation that there is no way to force false positive rate to be less than some constant. For example to say that the model would limit maximum 5% of all benign users. On the other hand we offer completely different approach of setting costs and that way our model computes the solution in which false positive rate is actually the best one in the given environment if all constants were set properly. To truly force some false positive rate user would have to run multiple setups and finally choose the one with the most satisfying false positive rate.

4.2 Blocking Model

4.2.1 Model Definition

Definition 4.2.1. Similarly as latency model we define a blocking model as an altered normal form game $\mathcal{B} = \langle N, A, u, n, D, C_b, C_m, P_m, P_b \rangle$ where:

- $N, n, D, C_b, C_m, P_m, P_b$ shares the same definition as in game G defined in Subsection 4.1.1.

- $A = A_a \times A_d$ is an action profile where A_i is an action set for player i. See notation in Equations (4.12).

$$A_a = [0,1]^n \tag{4.12a}$$

$$A_d = \{\, l \,|\, l : A_a \to \{0,1\} \,\} \tag{4.12b}$$

- $u = u_a \times u_d$ is an utility profile consisting of payoff functions $u_i : A \to \mathbb{R}$ for player i. Each player wants to maximize his payoff function. See definitions in Equations (4.13).

$$u_a(f,l) = + R(f) \cdot (1 - l(f)) \cdot P_m \cdot C_m \tag{4.13a}$$

$$\begin{aligned} u_d(f,l) = &- [R(f) \cdot (1 - l(f)) \cdot P_m \cdot C_m] \\ &- [(\sum_{d \in D} l(d) \cdot p(d)) \cdot P_b \cdot C_b] \end{aligned} \tag{4.13b}$$

$$R(f) = \prod_{j=1}^{n} f_j$$

where function $p : D \to [0,1]$ represents a probability distribution over features in provided benign data set D.

There are two major differences comparing to the previous latency model. Firstly the defender is now doing a classification instead of continuous regression, because a defender's action is now a binary function classifying each attacker's features either as malicious (1 = blocking) or benign (0 = no blocking). Secondly, there are no α_m and α_b constants, because they are no longer needed.

The utility functions are also similar as in previous model except that the attacker's payoff (or the defender's punishment) is also binary and the attacker either gets his outcome or he receives payoff equaled zero in case of classified as malicious.

4.2.2 Solving the Game

Using completely the same procedure as in deriving the game $\mathcal{G}'$ in Seubsection 4.1.4 we are allowed to infer strategically equivalent game $\mathcal{B}'$ from $\mathcal{B}$ with utility functions defined in Equations (4.14).

$$u_a(f,l) = +[R(f) \cdot (1 - l(f))] + (\sum_{d \in D} l(d) \cdot p(d)) \cdot C \tag{4.14a}$$

$$u_d(f,l) = -[R(f) \cdot (1 - l(f))] - (\sum_{d \in D} l(d) \cdot p(d)) \cdot C \tag{4.14b}$$

$$C = \frac{P_b \cdot C_b}{P_m \cdot C_m} \tag{4.14c}$$

We observe that game $\mathcal{B}'$ looks almost the same as game $\mathcal{G}'$ except that the defender's action sets differ. In $\mathcal{G}'$ the defender's actions classify

$$\begin{array}{rl} a_1 = & \begin{array}{|c|c|c|}\hline 1 & 0 & 1 \\ \hline 0 & 1 & 1 \\ \hline 0 & 0 & 0 \\ \hline \end{array} \\ \text{prob. } 30\% & \\[1em] a_2 = & \begin{array}{|c|c|c|}\hline 0 & 1 & 1 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array} \\ \text{prob. } 70\% & \end{array} \iff \begin{array}{c} a_3 = \\ \begin{array}{|c|c|c|}\hline 0.3 & 0.7 & 0.1 \\ \hline 0 & 0.3 & 0.3 \\ \hline 0 & 0 & 0 \\ \hline \end{array} \\ \text{prob. } 100\% \end{array}$$

Table 4.1: Equivalent transformation of mixed strategy to pure strategy

attacker's features continuously (latency in range between 0 and 1) but in game $\mathcal{B}'$ the defender's classification is binary.

Nonetheless, if we look in the Tables 4.1 (matrices represent classification of regions in a plain by defender's actions $a_1, a_2 \in A_d$), we can see that every defender's mixed strategy using binary classification may be transformed to a pure strategy if we allow continuous classification. The same way every pure strategy using continuous classification might be transformed to a mixed strategy of actions with binary classification.

The strategic equivalence (defined in Definition 3.1.6) is defined by exactly the same set of Nash Equilibria. Since we are able to model every binary classification strategy with continuous classification strategy and vice-versa without loss of generality, we are able to upgrade defender's actions to continuous classification. By applying this approach we obtain a new game which is strategically equivalent to the original one.

Definition 4.2.2. To sum this up, final altered blocking model $\mathcal{B}_f$ looks the same as $\mathcal{B}$ except that action profile A and payoff functions u, which can be seen in Equations (4.15); (4.16), slightly differ.

$$\begin{aligned} A_a &= [0,1]^n \\ A_d &= \{ \quad l \quad | \quad l : A_a \to [0,1] \} \end{aligned} \tag{4.15}$$

$$\begin{aligned} u_a(f,l) &= +[R(f)\cdot(1-l(f))] + (\sum_{d\in D} l(d)\cdot p(d))\cdot C \\ u_d(f,l) &= -[R(f)\cdot(1-l(f))] - (\sum_{d\in D} l(d)\cdot p(d))\cdot C \\ C &= \frac{P_b\cdot C_b}{P_m\cdot C_m} \end{aligned} \tag{4.16}$$

We managed to get the blocking model to the state where we may skip the elaboration about computing Nash equilibrium due to the reasons mentioned in following Subsection 4.2.3.

4.2.3 Discussion

We skipped the description of solving technique because we realize our final blocking model $\mathcal{B}_f$ defined in Definition 4.2.2 and final latency model $\mathcal{G}'$ defined in Definition 4.1.2 differ only in constants α_m and α_b. In fact our blocking model is a special case of game $\mathcal{G}'$ with constants $\alpha_m = \alpha_b = 1$. It means that our latency model finely generalizes the problem. That is why we did not want to literally rewrite the solving Subsection 4.1.4, because this approach might be applied in the exact same way to solve our blocking model.

Chapter 5

Experiments

5.1 Experiments Settings

All experiments were run on hardware with 4-cores at a 2.7GHz frequency, 8GB RAM and graphics processing unit Nvidia Tesla T4 using Google cloud platform. For implementation details, we refer the reader to see Appendix A.

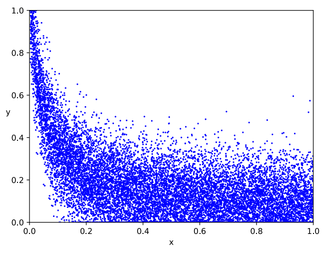
5.2 Influence of Best Response Quality on Double Oracle Convergence

We use double oracle algorithm, which assumes that the truly best responses are found in each iteration of the algorithm. However, since we are using neural networks to model the defender's actions, it gets challenging to train accurate best responses. That is why we want to examine how this fact limits us and see how the convergence of double oracle algorithm is changing upon searching for the defender's best responses with different quality.

The approach of how we manage to receive *worse* or *more accurate* best responses is primarily by adjusting a number of learning epochs during which we train each neural network. Hence, we wish to inspect how the convergence of double oracle applied to game $\mathcal{G}'$ is changing when using the defender's neural networks modeled with various learn-

$\alpha_b = 3$	$P_b = 0.99$	$C_b = 5 \cdot 10^2$	$\varepsilon_d = 10^{-3}$
$\alpha_m = 1$	$P_m = 10^{-2}$	$C_m = 1 \cdot 10^4$	$\varepsilon_a = 10^{-3}$

Table 5.1: Values of constants used in Experiments 5.2 and 5.3


(a) 2D synthetic distribution

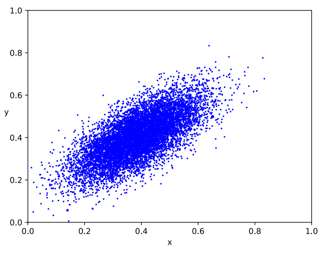

(b) 2D normal distribution

Figure 5.1: Generated data sets used in Experiments 5.2 and 5.3.

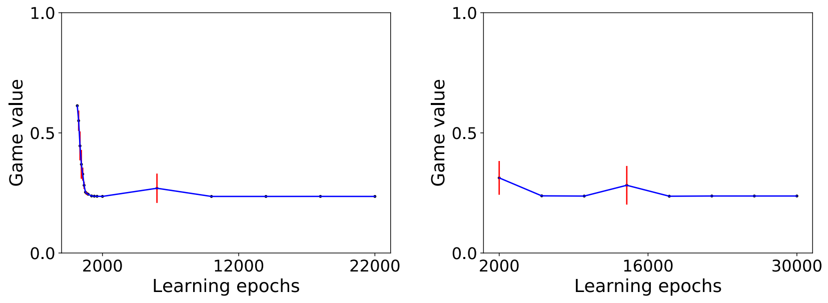
ing epochs. We start with 2000 learning epochs and increase to 30000 epochs by 4000 steps. We assume that neural networks trained with more learning epochs represent superior best responses. Also, we realize that different architecture of neural network substantially influences the process of learning. That is why we run this experiment for a shallow neural network with 5 neurons and a deep neural network with 35 neurons (the network with only a single hidden layer is conventionally called shallow, while the ones with two or more hidden layers are called deep).

For this experiment we use random constants values defined in Table 5.1. Even though values of these constants are not crucial for the development of this experiment. Finally, as data set representing benign users we use generated normal distribution visualized in Figure 5.1b.

As metrics for evaluating how the convergence of double oracle is changing we use a final value of the game and elapsed time until the game converges.

5.2.1 Results

Figures 5.2 show means with 95% confidence intervals of game value convergence and we can observe that in almost all setups we ended up



(a) The convergence of game value using deep neural network with 35 neurons

(b) The convergence of game value using shallow neural network with 5 neurons

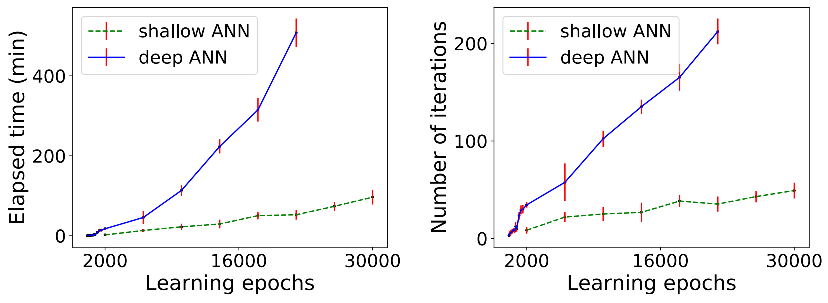
Figure 5.2: Convergence of game value using normal distribution data and various numbers of learning epochs

with exactly the same value. The result is pleasantly surprising because we see that we might actually use fewer learning epochs and still solve the game in a way as if we would utilize huge neural networks with many learning epochs and thus save a lot of computational time.

For comparison we may notice how elapsed time increases for big neural networks and many learning epochs in Figure 5.3a . Time the game takes to converge becomes actually crucial when approaching to many learning epochs as we can see that we did not even manage to compute all setups of learning epochs for deep neural network and rather stopped at 22000.

One might argue that the extra time consumption is obvious and is caused basically by extra time needed for neural network training phase. Of course there is an extra time needed to train each neural network but also double oracle algorithm takes extra time to converge as we might spot in Figure 5.3b, where we witness that likewise the iterations of double oracle increase.

We assume that the surprising result is caused by overfitting the best responses when using too many learning epochs. What it means is that the smart defender keeps finding best responses which classify exclusively current attacker's actions as malicious. That is why attacker has a lot



(a) Required time for double oracle algorithm to converge (b) Number of iterations double oracle needs to converge

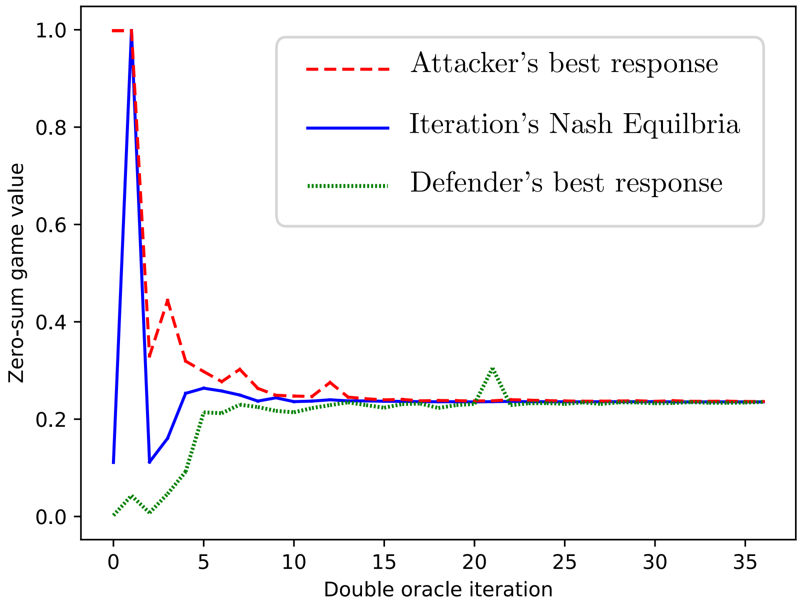
Figure 5.3: Duration of double oracle algorithm

free space where he is able to deviate and that is why it takes more iterations for double oracle algorithm to converge in order for defender to score all crucial spots with positive latency. Figures 5.4 showing the game value convergence within all iterations confirm our hypothesis. Alternatively in Figure 5.7 we are able to see some examples of best responses when training neural network with 22000 or 2000 epochs and differentiate overfitting when using 22000 learning epochs compared to 2000 epochs.

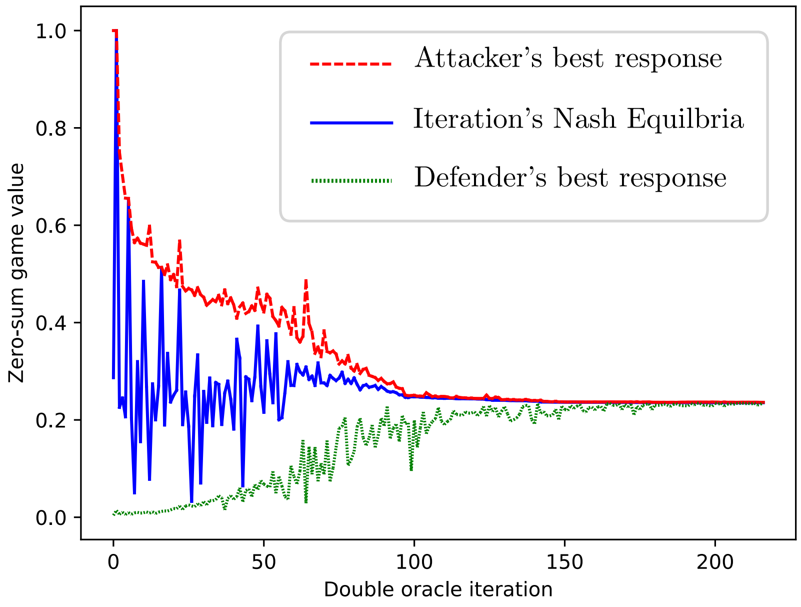
Another valid argument says that values ε_a and ε_d are set too big and that is the reason we cannot distinguish when using good best responses, because with lower values of ε_a, ε_d the bigger accuracy would be detectable. That might be true but we do not think that difference $\pm 10^{-3}$ in game value would be striking and causing superiorly better results.

Finally the origin might be embedded in our data set being extremely easy to learn. That is why we tried to run the same setup with more synthetic data shown in Figure 5.1a and received results visualized in Figure 5.8. We notice that shallow neural networks start having issues finding sufficient best responses, however the deep neural networks still have no problems even with 2000 learning epochs.

To sum this experiment up we unfortunately did not manage to run the experiment with very unlikely distributed data for which even deep neural network would fail having constant results. That is why we cannot



(a) Convergence of game using 2000 learning epochs



(b) Convergence of game using 22000 learning epochs

Figure 5.4: Convergence of game values within each iteration using deep neural network

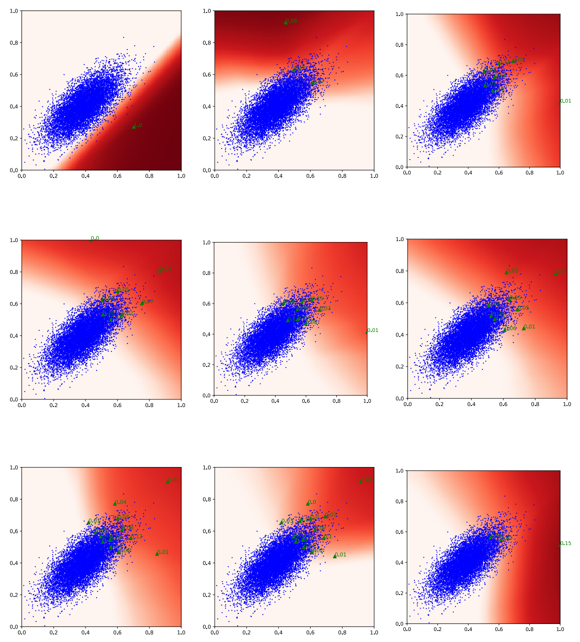


Figure 5.5: Sample of best responses using 2000 learning epochs

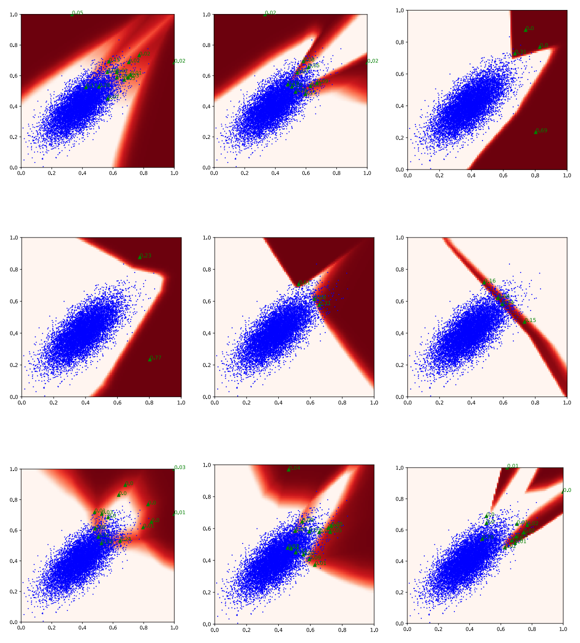


Figure 5.6: Sample of best responses using 22000 learning epochs

Figure 5.7: Visual example of best responses using either few or many learning epochs

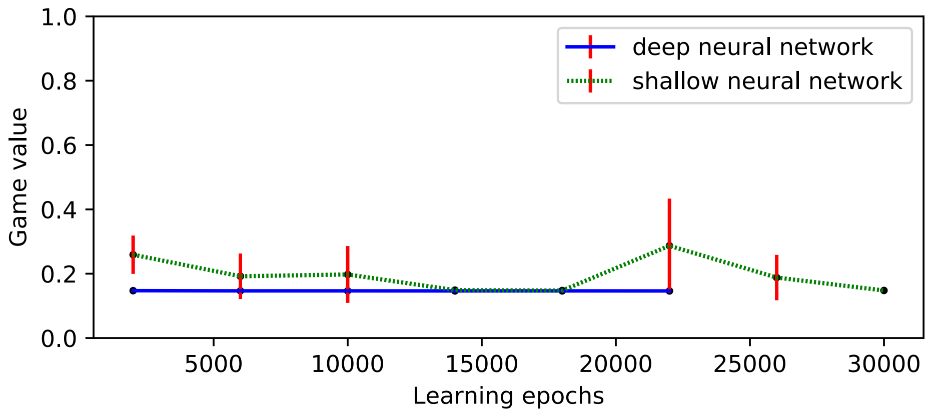


Figure 5.8: Convergence of game value using synthetic data and various numbers of learning epochs

conclude that similar results would happen for all data sets. We only show that the fact of not training truly best responses does not have to necessary influence correctness of the final result when using double oracle algorithm and that our 35 neurons neural network seems sufficient for all normal data sets.

5.3 Performance of Proposed Solving Methods

In this experiment we want to examine a scalability and an optimality of our double oracle solving technique applied on our proposed model. As metrics to evaluate this experiment we intend to use a final zero-sum game value and required time to converge. The plan is to inspect the performance for $n = \{2, 3\}$ using configuration in Table 5.1 and data set visualized in Figure 5.1b representing normal distribution.

In order to fully analyze the performance we wish to compare the final results with results received from application of discretized linear programming also mentioned in solving Subsection 4.1.4.

5.3.1 Results

Final results may be seen in Table 5.2. We observe that our double oracle technique stands perfectly in terms of optimality, because when we

dim.	**solver**	**Defender's** classification **latencies**	**Attacker's actions per one axis**	**Zero-sum game value**	**Elapsed time (min)**
$n = 2$	LP	$L = \{0, .25, .5, .75, 1\}$	$x = \{0..1\}$ $\|x\| = 50$	0.2448	0.99 $\pm$0.02
	DO	$L = [0, 1]$	$x = \{0..1\}$ $\|x\| = 50$	0.2354	56.38 $\pm$13.90
	DO	$L = [0, 1]$	$x = [0, 1]$	0.2353	48.71 $\pm$7.60
$n = 3$	LP	$L = \{0, .25, .5, .75, 1\}$	$x = \{0..1\}$ $\|x\| = 50$	N/A	> 1 day
	DO	$L = [0, 1]$	$x = \{0..1\}$ $\|x\| = 50$	N/A	> 1 day
	DO	$L = [0, 1]$	$x = [0, 1]$	0.16	149.59 $\pm$20.27

Table 5.2: Result of Experiment 5.3 examining scalability and optimality of our solving algorithms. *DO* stands for double oracle and *LP* stands for Linear Programming

discretize the attacker's actions the same way as in linear programming approach we receive lower zero-sum game value. It means the defender gets better utility, and thus modeling the defender's latency classification continuously with neural networks work fine.

Regarding the scalability we discover that the linear programming approach stands much better for $n = 2$. However, when we scale to $n = 3$ we did not even manage to compute the results for discretized attacker and had to stop the experiment setups after one day. One way would be to discretize less but we opine that in that case the results would not be accurate and thus not suitable for comparison. In this case we note that our solution scales much better and outputs more accurate results since both the defender's and the attacker's actions are modeled continuously.

5.4 Case Study - DNS Data Exfiltration

Finally, we would like to apply our model to some real world problem and see how it practically goes. We managed to obtain traffic records about the DNS requests within the CTU network. Thus, we try to focus on the DNS data exfiltration.

DNS stands for the Domain Name System and de-facto represents a phone book of the internet. An IP address is assigned to almost every host in the internet but IP addresses cannot be easily memorized so majority of people use hostnames instead and let the DNS to translate each hostname to its unique IP address.

Since DNS is fundamentally the core part of the internet functionality, the DNS requests are generally not regulated or blocked and that is why hackers might use these requests to smoothly exfiltrate private data from any network[17].

5.4.1 Settings

In order to run our model we need to first figure out values of constants for our game and compute features out of the DNS logs from the CTU network.

Constants are set according to the Table 5.3. We opine that if the attacker exfiltrates the data his gain decreases linearly with increasing latency since he is able to proportionally the latency transmit less data, that is why the constant α_m is set to one. We also think of the CTU network as not so important in providing such reliable service not to be able to increase a latency for the DNS requests of some benign users and that is the reason we set $\alpha_b = 2.5$. From the same reason we also set constant C_m much higher than C_b, because we think that the potential data breach would be more painful than increasing the latency for some benign users' DNS requests.

Lastly, we need to think of the features. Scoring each DNS request in real time would be computationally too expensive and also probably not accurate, because the attacker could exploit this fact and send many

$\alpha_b = 2.5$	$P_b = 0.999$	$C_b = 5 \cdot 10^4$	$\varepsilon_d = 10^{-3}$
$\alpha_m = 1$	$P_m = 1 \cdot 10^-3$	$C_m = 5 \cdot 10^6$	$\varepsilon_a = 10^{-3}$

Table 5.3: Values of constants for case study Experiment 5.4

smaller requests. That is why evaluating bigger time slots per each IP address seems more reasonable. We assume that each IP address belongs solely to either malicious or benign user. Choosing proper time slots might be topic for a discussion but for our experiment we use 5 seconds time slots and compute following features:

- **An average length** - we compute the average length as mean of lengths of all queries inside a given time slot divided by 255 (maximum length of DNS query).
- **An average entropy** - we compute the average entropy as mean of entropies of all queries inside a given time slot divided by 5.8 (a maximum entropy for the dns query given its maximum length and allowed characters).
- **A number of requests** - computing this feature is a little bit tricky since we need all features to be normalized between 0 and 1 but there is not any guarantee for maximum number of the DNS queries in the 5 seconds time slots. Thus, we decide by our point of view for the maximum to be 250 and value of this feature to be number of all the DNS request inside the given time slot divided by 250.

5.4.2 Results

Double oracle algorithm converged after 52 iterations and 73.5 minutes. Final ε-equilibrium produces 1.7‰ false positive rate which we acknowledge as a great result considering we do not have any fix constraint for precise false positive rate.

Unfortunately we are not allowed to publish any raw queries from the DNS logs. However, the most constrained benign time slot by final

average length	average entropy	number of requests	probability
25.5	3.6	94.35	28.1%
25.5	3.54	122.4	17.5%
38.25	3.24	66.3	15.9%
33.15	4.26	63.75	11.1%
140.25	3.84	22.95	6.7%
35	2.76	153.0	6.6%
25.5	3.36	155.55	4.2%
255	5.8	250	3.9%

Table 5.4: Denormalised attacker's final ε-equilbria actions with probability greater than 3% from DNS case study Experiment 5.4

ε-equilibrium is set of following features $(0.11, 0.60, 0.89)$ scored with latency 75%, which might seem like a lot but as our false positive rate says it is nothing compared to all other benign requests. Also this benign user still eventually receives his response, and that is where we see our most important contribution.

The attacker's most played actions from ε-equilibrium mixed strategy are shown denormalised in Table 5.4 where we observe that the attacker ends up sending many small requests with mostly big entropy.

5.4.3 Discussion

We found out that the fact of having all attacker's features normalised in the range from zero to one might complicate the application of our model when the maximum value of the features is questionable. Fortunately this limitation is solvable by just determining the ceiling; nonetheless, extra care must be taken when we compute the features.

Restating the results we observe that the proposed solving technique outputs optimal value and scales favourably. As a highlight of our experiments, we state the result of our first experiment. We managed to show that the double oracle algorithm does not necessarily need truly best responses in order to converge smoothly and find ε-equilibrium, which we perceive as a remarkable discovery for practical future usage of our model.

Chapter 6

Conclusion

In this thesis, we propose a general sum game between the defender and the attacker where we consider increasing latency as the defender's action. We manage to transform the game to zero sum game and propose solving technique using double oracle algorithm to compute ε-equilibrium. We experimentally show that modeling the defender's actions as artificial neural networks does not diminish the optimality of the final solution. In fact, the double oracle algorithm in conjunction with defender's neural networks produces more optimal results than applying linear programming approach to discretized game.

One of our most significant contributions inheres in having a generic solution which is able to model all kinds of specific use-cases by proper configuration. Another advantage is that our solution needs data set with only benign users because usually getting labeled data set with all used classes is the most challenging task. Also, our model ensures that even the adaptive attacker cannot shape his attacks to perform a better attack. The reason is that the game theory provides us such solution that no player is able to get more gain by deviating.

Overall the approach with slowing the attacker's traffic down seems like superiorly better approach than the blocking approach. However, the topic of increasing latency is quite vast and yet unexplored; thus, further work needs to be done. For example, the defender's actions do not have to be modeled using neural networks, and other approaches might be tested. Future work should also concentrate on modeling the solutions with increasing latency even for specific use-cases.

To sum our work up, we succeeded in fulfilling the goals of the thesis and propose a game-theoretic model and implementation, which experimentally show positive results. One of our future step to validate practical usage of our work is to apply the model to real traffic using for example modified routers.

Appendix A

Reference Implementation

A.1 Implementation Overview

The implementation was written in Python 3.7. For further details, we refer the reader to see the source code located in the attachments of the thesis. To run the source code locally, follow instructions written in a root folder in a file *readme.md.*

To implement the defender's actions, we used a machine learning library PyTorch. The architecture of each neural network consists of linear layers with ReLu activation functions. As a last activation function used in the output layer we used activation function called SoftClip introduced in [18]. The reason is that SoftClip looks similar to Sigmoid, but converges faster, so we are able to model even zero/one latency. For precise implementation of defender's actions see source code in file *src/neural_networks/network.py.*

A.2 List of Attachments

- A CD with the reference implementation. The real data set representing benign users within the CTU network is not submitted due to privacy reasons.

List of Used Symbols

Game Theory Generally

$A = A_1 \times A_2$	Action profile
A_i	Action set for player i
$BR(s_{-i})$	Set of player i's best responses given strategy profile s
$G = \{\, N, A, u \,\}$	Normal form game
N	A set of players within a game
$S = S_1 \times S_2$	Strategy profile
S_i	Set of all probability distributions over actions A_i
$u = u_1 \times u_2$	Profile of utility functions
$u_i : A \to \mathbb{R}$	Utility function for player i
$u_i : S \to \mathbb{R}$	Utility function taking strategy profile as input for player i

Models' Symbols

$\alpha_b \in \mathbb{R}^+$	Constant describing severity of increasing latency to benign user used in function h_b
$\alpha_m \in \mathbb{R}^+$	Constant describing how increasing latency affects the attacker used in function h_m
$\mathbb{X}$	Set of attacker's one-axis descretized actions

Θ	Discretized set $\Theta = \{\,\theta \mid \theta(x,l) \in \mathbb{R}^+$ determines probability of playing latency $l \in L$ against features $x \in \mathbb{X}^n\}$
ε_a	Constant determining minimal required improvement of attacker's best response payoff compared to Nash equilibria value to consider it as a new action
ε_d	Constant determining minimal required improvement of defender's best response payoff compared to Nash equilibria value to consider it as a new action
$C = \dfrac{P_b \cdot C_b}{P_m \cdot C_m}$	Constant unifying other constants
$C_b \in \mathbb{R}^+$	Cost for blocking all benign users (benign value)
$C_m \in \mathbb{R}^+$	Cost for letting attacker perform not detected attack in its full extent (malicious value)
$h_b : [0,1] \to \mathbb{R}$	Function describing how latency increases costs for false positive classifications
$h_m : [0,1] \to \mathbb{R}$	Function describing how latency affects attacker's gain
L	Set of discretized latencies available for defender's classification
$N = \{\,a, d\,\}$	Set of the players - the defender d and the attacker a
$n \in \mathbb{N}$	Dimension of attacker's actions/features
$P_b \in [0,1]$	Value representing portion of benign requests in a certain time slot compared to the malicious requests
$P_m \in [0,1]$	Value representing portion of malicious requests in a certain time slot compared to the benign requests

Bibliography

[1] J. v. Neumann. “Zur Theorie der Gesellschaftsspiele”. In: *Mathematische Annalen* 100.1 (Dec. 1928), pp. 295–320. ISSN: 1432-1807. DOI: 10.1007/BF01448847.

[2] John Nash. “Non-Cooperative Games”. In: *Annals of Mathematics* 54.2 (1951), pp. 286–295. ISSN: 0003486X.

[3] “A Polynomial-Time Algorithm for Solving Linear Programs”. In: *Math. Oper. Res.* 5.1 (Feb. 1980), pp. iv–iv. ISSN: 0364-765X. DOI: 10.1287/moor.5.1.iv.

[4] George Cybenko. “Approximation by superpositions of a sigmoidal function”. In: *MCSS* 2 (1989), pp. 303–314.

[5] Y. Nakamori and M. Ryoke. “Identification of fuzzy prediction models through hyperellipsoidal clustering”. In: *IEEE Transactions on Systems, Man, and Cybernetics* 24.8 (Aug. 1994), pp. 1153–1173. ISSN: 0018-9472. DOI: 10.1109/21.299699.

[6] Corinna Cortes and Vladimir Vapnik. “Support-Vector Networks”. In: *Machine Learning* 20.3 (Sept. 1995), pp. 273–297. ISSN: 1573-0565. DOI: 10.1023/A:1022627411411.

[7] Luchan Liu. “The Invariance of Best Reply Correspondences in Two-Player Games”. In: (1996).

[8] H Brendan McMahan, Geoffrey J. Gordon, and Avrim Blum. “Planning in the Presence of Cost Functions Controlled by an Adversary.” In: Jan. 2003, pp. 536–543.

[9] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization.* Cambridge University Press, 2004. DOI: 10.1017/CBO9780511804441.

[10] S. Chen and Y. Tang. “Slowing down Internet worms”. In: *24th International Conference on Distributed Computing Systems, 2004. Proceedings.* Mar. 2004, pp. 312–319. DOI: 10.1109/ICDCS.2004.1281596.

[11] Xi Chen and Xiaotie Deng. "Settling the Complexity of 2Player Nash-Equilibrium". In: *Electronic Colloquium on Computational Complexity - ECCC* (Jan. 2005).

[12] Cynthia Wong et al. "Empirical Analysis of Rate Limiting Mechanisms". In: *Recent Advances in Intrusion Detection*. Ed. by Alfonso Valdes and Diego Zamboni. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 22–42. ISBN: 978-3-540-31779-1.

[13] Yoav Shoham and Kevin Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations.* New York, NY, USA: Cambridge University Press, 2008. ISBN: 0521899435.

[14] Leonid Kalinichenko, I Shanin, and I Taraban. "Methods for anomaly detection: A survey". In: *CEUR Workshop Proceedings* 1297 (Jan. 2014), pp. 20–25.

[15] Mohiuddin Ahmed, Abdun Naser Mahmood, and Jiankun Hu. "A survey of network anomaly detection techniques". In: *Journal of Network and Computer Applications* 60 (2016), pp. 19–31. ISSN: 1084-8045.

[16] L. Dritsoula, P. Loiseau, and J. Musacchio. "A Game-Theoretic Analysis of Adversarial Classification". In: *IEEE Transactions on Information Forensics and Security* 12.12 (Dec. 2017), pp. 3094–3109. ISSN: 1556-6013. DOI: 10.1109/TIFS.2017.2718494.

[17] Asaf Nadler, Avi Aminov, and Asaf Shabtai. "Detection of Malicious and Low Throughput Data Exfiltration Over the DNS Protocol". In: *CoRR* abs/1709.08395 (2017). arXiv: 1709.08395.

[18] Matthew D. Klimek and Maxim Perelstein. "Neural Network-Based Approach to Phase Space Integration". In: (Oct. 2018).