Finding Effective Security Strategies through Reinforcement Learning and Self-Play

Kim Hammar [†‡] and Rolf Stadler[†‡]
† Division of Network and Systems Engineering, KTH Royal Institute of Technology, Sweden
‡ KTH Center for Cyber Defense and Information Security, Sweden
Email: {kimham, stadler}@kth.se

Abstract—We present a method to automatically find security strategies for the use case of intrusion prevention. Following this method, we model the interaction between an attacker and a defender as a Markov game and let attack and defense strategies evolve through reinforcement learning and self-play without human intervention. Using a simple infrastructure configuration, we demonstrate that effective security strategies can emerge from self-play. This shows that self-play, which has been applied in other domains with great success, can be effective in the context of network security. Inspection of the converged policies show that the emerged policies reflect common-sense knowledge and are similar to strategies of humans. Moreover, we address known challenges of reinforcement learning in this domain and present an approach that uses function approximation, an opponent pool, and an autoregressive policy representation. Through evaluations we show that our method is superior to two baseline methods but that policy convergence in self-play remains a challenge.

Index Terms—Network Security, Reinforcement Learning, Markov Security Games

I. INTRODUCTION

An organization's security strategy has traditionally been defined, implemented, and updated by domain experts. Although this approach can provide basic security for an organization's communication and computing infrastructure, a growing concern is that update cycles become shorter and security requirements change faster [1]. Two factors contribute to this development: on the one hand, infrastructures keep changing due to functional upgrades and innovation, and, on the other hand, attacks increase in sophistication. As a response, significant efforts are made to *automate security processes and functions*, including automated threat modeling [2], [3], game-theoretic approaches [4], [5], evolutionary methods [6], [7], and learning-based solutions [8], [9].

To automate the response to a cyber attack in a changing environment, we view the interaction between an attacker and a defender as a game, and model the *evolution of attack and defense strategies* with automatic self-play. To avoid the need for domain knowledge to be coded into agent strategies, we study methods where the strategies evolve without human intervention from random configurations. In particular, we use self-play, a setting where both the attacker and the defender learn by interacting with each other, as a technique to automatically harden the defense.

Formally, we model the interaction between an attacker and a defender as a Markov game. We then use simulations of self-play where autonomous agents interact and continuously update their strategies based on experience from previously played games. To automatically update strategies in the game, several methods can be used, including computational game theory, evolutionary algorithms, and reinforcement learning. In this work, we choose reinforcement learning, which allows us to use simulations of self-play where both agents evolve their strategies simultaneously.

In pursuing this approach, we face several challenges. First, from a game-theoretic perspective, the game between an attacker and a defender is one with partial observability. That is, a defender can typically not observe the state and location of an attack, and an attacker is generally not aware of the full state of a defender's infrastructure. Second, due to the scale and complexity of computer infrastructures, the games that we consider have large state and action spaces, which make them hard to analyze theoretically. Third, as both the attacker and the defender learn simultaneously during self-play, the reinforcement learning environment becomes non-stationary [10], which is a known challenge for reinforcement learning and often results in agent strategies that do not converge [11], [12], [13].

We develop our method around the use case of *intrusion prevention*. This use case considers a defender that owns an infrastructure that consists of a set of connected components (e.g. a communication network), and an attacker that seeks to intrude on the infrastructure. The defender's objective is to protect the infrastructure from intrusions by monitoring the network and patching vulnerabilities. Conversely, the attacker's goal is to compromise the infrastructure and gain access to a critical component. To achieve this, the attacker must explore the infrastructure through reconnaissance and attack components along the path to the critical component. In this work, we study a relatively small and simple infrastructure of the defender, which allows us to easier compare security strategies and focus on the principles of our method.

With this paper, we make the following contributions. First, we show, using a small infrastructure, that reinforcement learning and self-play can be used to automatically discover effective strategies for intrusion prevention. Although self-play has been used successfully in other domains, such as board games [12] and video games [14], [13], our work is, to the best of our knowledge, the first study of self-play in the context of network and systems security. Second, we present a reinforcement learning method to address the challenges that appear

in the context of our use case and show through evaluations that it outperforms two baselines. Specifically, to tackle the problem of the high dimensional state space, we use function approximation implemented with neural networks. To address the non-stationary reinforcement learning environment, we use an opponent-pool technique, and to deal with the large action space, we propose an autoregressive policy representation.

The remainder of this paper is structured as follows. We first cover the necessary background information on partially observed Markov decision processes, reinforcement learning, and Markov games. Then we introduce a game model for intrusion prevention and present our method together with experimental results. Lastly, we describe how our work relates to prior research, present our conclusions, and provide directions for future work.

II. POMDPS, REINFORCEMENT LEARNING AND MARKOV GAMES

This section contains background information on Markov decision processes, reinforcement learning, and Markov games.

A. Markov Decision Processes

A Markov Decision Process (MDP) is defined by a six-tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}_{ss'}^a, \mathcal{R}_{ss'}^a, \gamma, \rho_0 \rangle$ [15]. $\mathcal{S}$ denotes the set of states and $\mathcal{A}$ denotes the set of actions. $\mathcal{P}_{ss'}^a$ refers to the transition probability of moving from state s to state s' when taking action a (Eq. 1), which has the Markov property (Eq. 2). Similarly, $\mathcal{R}_{ss'}^a \in \mathbb{R}$ is the expected reward when taking action a in state s to transition to state s' (Eq. 3). Finally, $\gamma \in (0, 1]$ is the discount factor and $\rho_0 : \mathcal{S} \mapsto [0, 1]$ is the initial state distribution.

$$\mathcal{P}_{ss'}^a = \mathbb{P}\left[s_{t+1} = s' | s_t = s, a_t = a\right] \quad (1)$$

$$\mathbb{P}\left[s_{t+1}|s_t\right] = \mathbb{P}\left[s_{t+1}|s_1, \ldots, s_t\right] \quad (2)$$

$$\mathcal{R}_{ss'}^a = \mathbb{E}\left[r_{t+1}|a_t = a, s_t = s, s_{t+1} = s'\right] \quad (3)$$

A Partially Observed Markov Decision Process (POMDP) is an extension of an MDP [16], [17]. The difference compared with an MDP is that, in a POMDP, the agent does not directly observe the states $s \in \mathcal{S}$. Instead, the agent observes observations $o \in \mathcal{O}$, which depend on the state s as defined by an observation function $\mathcal{Z}$. Specifically, a POMDP is defined by an eight-tuple $\mathcal{M}_\mathcal{P} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}_{ss'}^a, \mathcal{R}_{ss'}^a, \gamma, \rho_0, \mathcal{O}, \mathcal{Z} \rangle$. The first six elements define an MDP. $\mathcal{O}$ denotes the set of observations and $\mathcal{Z} = \mathbb{P}[o|s]$ denotes the observation function, where $o \in \mathcal{O}$ and $s \in \mathcal{S}$.

B. The Reinforcement Learning Problem

The reinforcement learning problem can be formulated as that of finding a policy π^* that maximizes the expected cumulative reward of an MDP over a finite horizon T [18]:

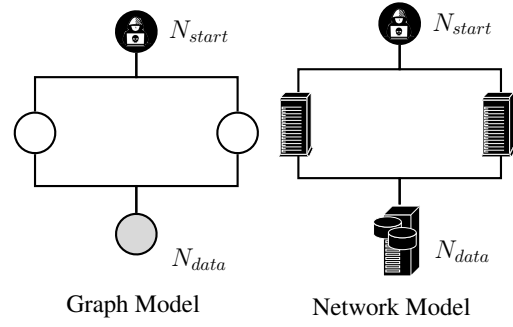$$\pi^* = \arg\max_\pi \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r_{t+1}\right] \quad (4)$$

Fig. 1: Modeling intrusion prevention as a Markov game.

The Bellman equations [19] (Eq. 5-6) relates the optimal policy π^* to the states and the actions of the MDP, where $\pi^*(s) = \arg\max_a Q^*(s, a)$.

$$V^*(s) = \max_a \mathbb{E}\left[r_{t+1} + \gamma V^*(s_{t+1})|s_t = s, a_t = a\right] \quad (5)$$

$$Q^*(s, a) = \mathbb{E}[r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a')|s_t = s, a_t = a] \quad (6)$$

A reinforcement learning *algorithm* is a procedure to compute or approximate π^*. Three main classes of such algorithms exist: value-based algorithms (e.g. Q-learning [20]), policy-based algorithms (e.g. PPO [21]), and model-based algorithms (e.g. Dyna-Q [18]).

C. Markov Games

Markov games, also known as stochastic games [22], provide a theoretical framework for *multi-agent* reinforcement learning [23]. A Markov game of N agents is defined as a tuple $\mathcal{M}_G = \langle \mathcal{S}, \mathcal{A}_1, \ldots, \mathcal{A}_N, \mathcal{T}, \mathcal{R}_1, \ldots \mathcal{R}_N, \gamma, \rho_0 \rangle$. S denotes the set of states and the list $\mathcal{A}_1, \ldots, \mathcal{A}_N$ denotes the sets of actions of agents $1, \ldots, N$. The transition function $\mathcal{T}$ is an operator on the set of states and the combined action space of all agents, $\mathcal{T} : \mathcal{S} \times \mathcal{A}_1 \times \ldots \times \mathcal{A}_N \mapsto \mathcal{S}$. Similarly, the functions $\mathcal{R}_1, \ldots \mathcal{R}_N$ define the rewards for agent $1, \ldots, N$ where $\mathcal{R}_i : \mathcal{S} \times \mathcal{A}_i \mapsto \mathbb{R}$. Moreover, γ is the discount factor and $\rho_0 : \mathcal{S} \mapsto [0, 1]$ is the initial state distribution. Finally, in the partially observed setting of a Markov game, the model also includes sets of observations $\mathcal{O}_1, \ldots, \mathcal{O}_N$ as well as observation functions $\mathcal{Z}_1, \ldots, \mathcal{Z}_N$ [24].

III. MODELING INTRUSION PREVENTION AS A MARKOV GAME

In this section, we model the use case of intrusion prevention as a zero-sum Markov game that involves two agents, an attacker and a defender.

A. Intrusion Prevention as a Game

The right side of Fig. 1 shows the infrastructure underlying our use case. It is depicted as a graph that includes four network components. The component N_{start} represents the attacker's computer and the remaining components represent the defender's infrastructure, where N_{data} is the component that the attacker wants to compromise.
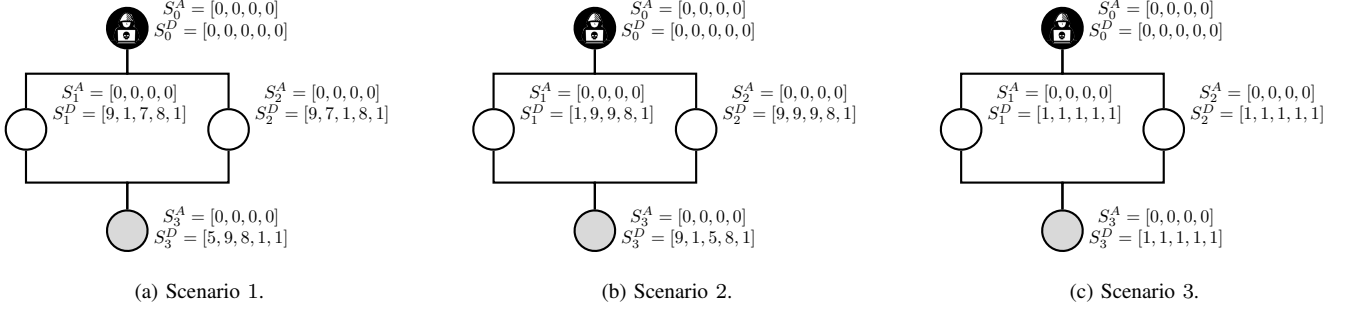
Fig. 2: Three scenarios of the intrusion prevention game. Scenarios 1 and 2 model infrastructures with strong defenses (first four defense attributes) but weak detection capabilities (fifth defense attribute). In scenario 1, each node contains one vulnerability (a defense attribute with a value ≤ 1), whereas in scenario 2 only one of the intermediary nodes has a vulnerability (the left one). Scenario 3 models an infrastructure with both weak defenses and weak detection capabilities.

To compromise the target component N_{data}, the attacker must explore the infrastructure through reconnaissance and compromise components on the path to N_{data}. At the same time, the defender monitors the network and increases its defenses to prevent the attacker from reaching N_{data}. In this adversarial process, both the attacker and the defender have a partial view of the network. At the beginning of the game, the attacker knows neither the topology of the infrastructure nor its vulnerabilities. In contrast, the defender has complete knowledge of the network's topology and the vulnerabilities of its components, but cannot observe the status of attacks.

The described adversarial process evolves as a round-based game. In each round, the attacker and the defender perform actions on components in the network, continuing until either the attacker wins the game by compromising N_{data}, or the defender wins the game by detecting the attacker.

B. A Formal Game Model of Intrusion Prevention

We extend prior work by others [25] and model the network infrastructure as a graph $\mathcal{G} = \langle \mathcal{N}, \mathcal{E} \rangle$ (see Fig. 1 left side). The nodes $\mathcal{N}$ of the graph represent the components of the infrastructure. Similarly, the edges $\mathcal{E}$ of the graph represent connectivity between components. The graph has two special nodes, N_{start} and N_{data}, representing the attacker's starting position and target, respectively. Moreover, each node $N_k \in \mathcal{N}$ has an associated node state, $S_k = \langle S_k^A, S_k^D \rangle$. The node's defense status S_k^D is only visible to the defender and consists of $m + 1$ attributes, $S_{k,1}^D \ldots, S_{k,m+1}^D$. Likewise, the node's attack status S_k^A is only visible to the attacker and consists of m attributes, $S_{k,1}^A, \ldots S_{k,m}^A$. The attribute values are natural numbers in the range $0, \ldots, w$, where $w > 0$.

The values of the attack attributes $S_{k,1}^A, \ldots, S_{k,m}^A$ represent the strength of m different types of attacks against node N_k, e.g. denial of service attacks, cross-site scripting attacks, etc. Similarly, the values of the defense attributes $S_{k,1}^D, \ldots, S_{k,m}^D$ represent the strength of the node's defenses against the m attack types and encodes the node's security mechanisms, such as firewalls and encryption functions. Additionally, each node N_k has a special defense attribute $S_{k,m+1}^D$, whose value represents the node's capability to detect an attack.

As an example, installing an intrusion detection system—a monitoring mechanism—at node N_k would correspond to an increase of the detection value $S_{k,m+1}^D$ in the model. Correspondingly, installing a rate-limiting mechanism—a security mechanism against denial of service attacks—at node N_k would correspond to an increase of the defense attribute $S_{k,j}^D$, where j is the attack type for denial of service attacks in the model.

The attacker can perform two types of actions on a visible node N_k: a reconnaissance action, which renders the defense status S_k^D visible to the attacker, or an attack of type $j \in 1 \ldots m$, which increases the attack value $S_{k,j}^A$ by 1. If the attack value exceeds the node's defense value, i.e. $S_{k,j}^A > S_{k,j}^D$, the attacker has compromised the node and the node's neighbors become visible to the attacker. Conversely, if the attacker performs an attack that does not compromise a node, the attack is detected by the defender with probability $p = \frac{S_{k,m+1}^D}{w+1}$, defined by the node's detection capability $S_{k,m+1}^D$.

Just like the attacker, the defender can perform two types of actions on node N_k: either a monitoring action, that improves the detection capability of the node and increments the detection value $S_{k,m+1}^D$, or a defensive action, that improves the defense against attacks of type $j \in 1 \ldots m$ and increments the defense value $S_{k,j}^D$.

The actions are performed on a round-by-round basis in the game. In each round, the attacker and the defender perform one action each, which brings the system into a new state. The game ends either when the attacker compromises the target node, N_{data} (attacker wins), or when the attacker is detected (defender wins). The winner of a game is rewarded with a utility of $+1$ whereas the opponent receives a utility of -1.

The game evolves as a stochastic game with the Markov property, $\mathbb{P}[s_{t+1}|s_t] = \mathbb{P}[s_{t+1}|s_1, \ldots, s_t]$, which follows trivially from the game dynamics defined above. The size of the state space $\mathcal{S}$ of the Markov game is $(w+1)^{|\mathcal{N}| \cdot m \cdot (m+1)}$. Finally, the size of the action spaces $\mathcal{A}_1$ and $\mathcal{A}_2$ for both the

attacker and the defender is $|\mathcal{N}| \cdot (m + 1)$.

C. Scenarios for the Intrusion Prevention Game

The Markov game defined above is parameterized by the graph's topology, the number of attack types m, the maximal attribute value w, and the initial node states $S_1, \ldots, S_{|\mathcal{N}|}$. In the remainder of the paper we focus on three specific scenarios that are depicted in Fig. 2. We define the number of attack types to be four ($m = 4$), the maximal attribute value $w = 9$, and initialize all attack states to zero: $S_{k,i}^A = 0$ for $k \in 1 \ldots |\mathcal{N}|$ and $i \in 1 \ldots m$. The three scenarios differ in how their defense states $S_{k,i}^D$ for $k \in 1 \ldots |\mathcal{N}|$ and $i \in 1 \ldots m + 1$ are initialized, as is illustrated in Fig. 2. Specifically, scenario 1 and 2 model infrastructures with strong defenses but weak detection capabilities whereas scenario 3 models an infrastructure with both weak defenses and weak detection capabilities.

IV. FINDING SECURITY STRATEGIES THROUGH REINFORCEMENT LEARNING

This section describes our method for finding strategies for the attacker and the defender in the game defined in Section III. Specifically, we describe how self-play in combination with reinforcement learning can be used to learn strategies without prior knowledge. We also discuss our approach to address the difficulties associated with the high-dimensional state and action spaces, as well as the challenges that come with a non-stationary environment.

A. Learning Policies through Self-Play

To simulate games using the model defined in Section III, actions for the attacker and the defender are sampled from policies[1]. In our approach, these policies are not defined by experts, but evolve through the process of self-play, which we implement as follows. First, we initialize both the attacker and the defender with random policies. Next, we run a series of simulations where the agents play against each other using their current policies. After a given number of games, we use the game outcomes and the game trajectories to update the policies using reinforcement learning. This process of playing games and updating the agents' policies continues until both policies sufficiently converge. Although self-play often converges in practice, as reported in [26], [12], [13], no formal guarantees for policy convergence in self-play have been established.

The Challenge of a Non-Stationary Environment: One reason for the lack of convergence guarantees of self-play is that the environment is non-stationary, which is an inherent challenge for reinforcement learning [10], [11]. Specifically, when multiple agents learn simultaneously in self-play, the agents' policies are part of the environment. This means that for each agent, the environment changes when the opponent updates its policy. Hence, the environment is non-stationary. To improve the possibility of convergence in self-play, despite

the non-stationary environment, different methods have been proposed in prior work [12], [13]. We describe our approach to this issue in Section IV-E.

B. Reinforcement Learning Algorithm

To learn strategies for the attacker and the defender in the game, we use the well-known reinforcement learning algorithms REINFORCE [27] and PPO [21]. Both of them implement the *policy gradient* method for solving the reinforcement learning problem (Eq. 4). This method can be summarized as follows. First, the policy π is represented as a parameterized function π_θ, where $\theta \in \mathbb{R}^d$. Second, an objective function $J(\theta)$ that estimates the expected reward for the policy π_θ is introduced. For example, the expected reward following policy π_θ can be estimated by sampling from the environment, e.g. $J(\theta) = \mathbb{E}_{o \sim \rho^{\pi_\theta}, a \sim \pi_\theta}[R]$, where ρ^{π_θ} is the observation distribution and R is the cumulative episode reward. Third, the optimization problem of maximizing $J(\theta)$ is solved using stochastic gradient ascent with a variant of the following gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{o \sim \rho^{\pi_\theta}, a \sim \pi_\theta} \left[\underbrace{\nabla_\theta \log \pi_\theta(a|o)}_{\text{actor}} \underbrace{A^{\pi_\theta}(o, a)}_{\text{critic}} \right] \quad (7)$$

where $A^{\pi_\theta}(o, a) = Q^{\pi_\theta}(o, a) - V^{\pi_\theta}(o)$ is the advantage function, that gives an estimate of how advantageous the action a is when following policy π_θ compared to the average action.

C. Function Approximation

The Markov game described in Section III has a high-dimensional state space where the number of states grows exponentially with the number of nodes $|\mathcal{N}|$ and attack types m. Specifically, as each node $N_k \in \mathcal{N}$ has m attack attributes and $m + 1$ defense attributes, whose values range from $0, \ldots, w$, the state space has size $|\mathcal{S}| = (w + 1)^{|\mathcal{N}| \cdot m \cdot (m+1)}$. This means that tabular reinforcement learning methods that rely on enumerating the entire state space are impractical. Instead, we use a parameterized function π_θ as a more compact representation of the policy than a table. In particular, we implement π_θ with a deep neural network that has a fixed set of parameters $\theta \in \mathbb{R}^d$. The neural network takes as input an observation o, and outputs a conditional probability distribution $\pi_\theta(a|o)$ over all possible actions a. Moreover, the neural network uses an actor-critic architecture [28] and computes a second output that estimates the value function $V^{\pi_\theta}(o)$, which is used to estimate $A^{\pi_\theta}(o, a)$ in the gradient in Eq. 7.

D. Autoregressive Policy Representation

Using reinforcement learning in environments with large action spaces is challenging and often results in slow convergence of policies [14]. In our case, the action space of the Markov game from Section III grows linearly with the number of nodes $|\mathcal{N}|$ and the number of attack types m. To deal with this challenge, we leverage the structure of the problem domain to reduce the size of the action space. Specifically,

---

[1]As we build on prior work in both reinforcement learning and game theory we use the terms "strategy" and "policy" interchangeably.
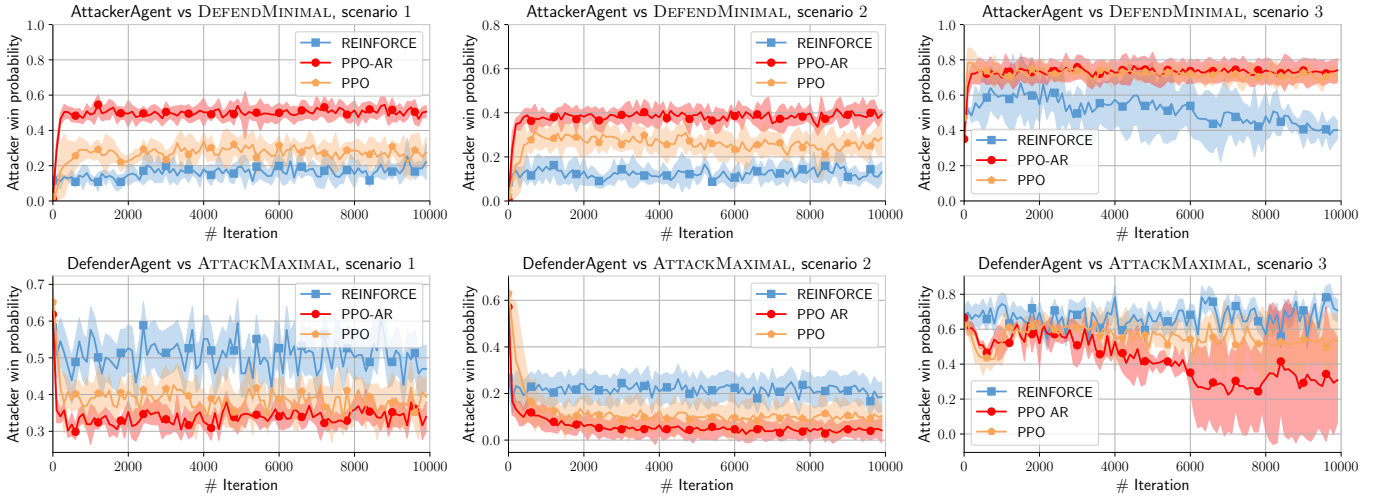
Fig. 3: Attacker win ratio against the number of training iterations; the top row shows the results from training the attacker against DEFENDMINIMAL; the bottom row shows the results from training the defender against ATTACKMAXIMAL; the three columns represent the three scenarios; the curve labeled PPO-AR shows the mean values of our proposed method; the results are averages over five training runs with different random seeds; the shaded regions show the standard deviation.

we represent an action as a sequence of two sub-actions: (1) select the node n to attack or defend; and (2) select the type of attack or defense a. As a consequence, we can decompose the policy $\pi(a, n|o)$ in two sub-policies, $\pi(a|n, o)$ and $\pi(n|o)$ based on the chain rule of probability (Eq. 8). This reduces the size of the action space of both the attacker and the defender from the size $|\mathcal{N}| \cdot (m + 1)$ to $|\mathcal{N}| + (m + 1)$.

$$\pi(a, n|o) = \pi(a|n, o) \cdot \pi(n|o) \qquad (8)$$

We implement each sub-policy with a neural network. First, $\pi_\theta(n|o)$ takes an observation o as input and decides the node n to attack or defend. Next, the second neural network $\pi_\phi(a|n, o)$ focuses on the selected node n to identify vulnerabilities that can be exploited or patched with an attack or defense of type a. Hence, the action is sampled in an *autoregressive* manner from the two sub-policies.

E. Opponent Pool

Although prior works have demonstrated that self-play can be a powerful learning method [12], [13], it is also well-known that self-play can suffer from training instability, which prevents the agents' policies from converging [29], [12], [13]. Specifically, it can occur that an agent adjusts its policy to its current opponent to such a degree that the policy overfits and becomes ineffective against other policies. Our approach to mitigate this problem is to sample opponent policies from a *pool* of policies during training [12], [13]. The opponent pool increases the diversity of policies, and as a result, reduces the chance of overfitting. To populate the pool, we add the agents' current attack and defense policies periodically to the pool during training.

V. LEARNING EFFECTIVE SECURITY STRATEGIES

In this section, we evaluate the method presented in Section IV for finding security strategies for the use case of intrusion prevention. We train reinforcement learning agents using simulations of self-play and evaluate our method with respect to convergence of agent policies. We also compare its performance with those of baseline reinforcement learning algorithms and inspect the learned policies. In sub-section V-A we train the attacker and the defender against static opponents, and in sub-section V-B we train both the attacker and the defender simultaneously.

Our implementation of the game model and the agents' policies is open source and available at `https://github.com/Limmen/gym-idsgame`. The simulations have been conducted using a Tesla P100 GPU and hyperparameters are listed in Appendix A.

A. Learning Policies Against a Static Opponent

We first examine whether our method can discover effective attack and defense policies against a *static* opponent policy. This means that we keep one policy fixed in self-play and that the other agent learns its policy against a static opponent. As one agent's policy is fixed, the environment for the other agent is stationary, which simplifies policy convergence.

1) Static Opponent Policies: We investigate two cases of learning against a static opponent. In the first case, the attacker learns its policy by playing against a static defender policy called DEFENDMINIMAL. In the second case, the defender learns its policy by playing against a static attacker policy called ATTACKMAXIMAL. DEFENDMINIMAL is a heuristic policy that updates the attribute with the minimal defense value across all nodes. Similarly, ATTACKMAXIMAL is a heuristic policy that updates the attribute with the maximal attack value
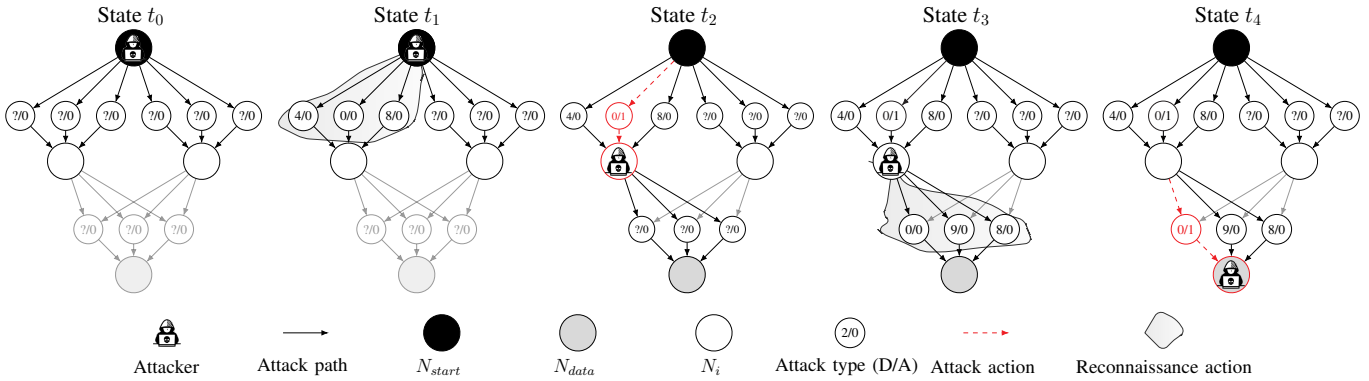
Fig. 4: An illustration of a learned attack strategy, evolving from left to right. The attacker first scans a neighboring node for vulnerabilities (low defense attributes) (state t_1). The attacker then exploits the found vulnerability (state t_2), compromises the node, and scans the target node N_{data} (state t_3). Finally, the attacker completes the intrusion by attacking N_{data} (state t_4).

across all nodes that are visible to the attacker. If several minimal or maximal values exists, DEFENDMINIMAL and ATTACKMAXIMAL picks the attribute to update at random.

2) Simulation Setup: To evaluate our method, we run simulations of the three scenarios described in Section III-C. Before each simulation run, we initialize the model with a random permutation of the defense attributes to prevent overfitting (see Fig. 2). In addition to the three scenarios, we evaluate three algorithms: our proposed method, PPO-AR, that uses an autoregressive policy representation (see Section IV-D), and two baseline methods, regular PPO and REINFORCE. Apart from the different algorithms and policy representations, all three methods use the same training setup and hyperparameters (Appendix A). For each scenario and algorithm, we run $10,000$ training iterations, each iteration consisting of $2,000$ game rounds. After each training iteration, the non-static agent updates its policy. Moreover, to measure stability we run each simulation five times with different random seeds, where a single simulation run accounts for 4 hours of training time on a P100 GPU. The average results are shown in Fig. 3.

3) Analysis of the results in Fig. 3: The top row of Fig. 3 shows the results from training the attacker against the static defender, and the bottom row shows the results from training the defender against the static attacker. The three columns represent the three scenarios. The x-axes denote the training iterations and the y-axes denote the attacker's empirical win ratio, calculated as the win ratio for the attacker in 100 evaluation games.

The top row of Fig. 3 demonstrates that the attacker quickly adjusts its policy to increase its win ratio. Specifically, we can observe that the policies converge after about 500 iterations. Furthermore, we can see that our method, PPO-AR, achieves a higher win ratio than the two baselines in all three scenarios. It can also be seen that among the baselines, PPO outperforms REINFORCE.

Looking at the bottom row of Fig. 3, we observe, similar to the top row, that the defender's policy converges quickly, although the results tend to have a higher variance. Moreover,

we see that our proposed method, PPO-AR, outperforms the baselines and achieves the lowest average win ratio of the attacker for all three scenarios. We also observe that PPO outperforms REINFORCE.

That PPO-AR outperforms the baselines indicates that the reduced action space that results from the autoregressive policy representation simplifies exploration, enabling the agents to discover more effective policies. Moreover, although we expected that PPO would do better than REINFORCE, as it uses a more theoretically justified policy update [21], it is surprising that the difference is so substantial in the results; we anticipate that it could be due to wrong setting of hyperparameters.

In the three columns of Fig. 3 we see that the similarities and the differences among the three scenarios (see Section III-C) are visible in the results. Specifically, that the win ratio of the attacker is lower in scenario 1 and 2 than in scenario 3 reflects the fact that scenario 1 and 2 have strong initial defenses, whereas scenario 3 has weak initial defenses. Similarly, that scenario 1 and scenario 2 have alike defenses can be seen on the similar curves in the two leftmost columns of Fig. 3. Moreover, the weak initial defense of scenario 3 can be seen in the high variance of our method in scenario 3 (Fig. 3, bottom row, right column). As all nodes are equally vulnerable in scenario 3, the static attacker can select any attack path. Consequently, the defender learns a defense policy that predicts a diverse set of attacks, leading to a high variance.

4) Inspection of Learned Policies: We find that the learned *attack* policies are deterministic and generally consist of two steps: (1) perform reconnaissance to collect information about neighboring nodes; and (2), exploit identified vulnerabilities (Fig. 4). Furthermore, we find that the learned *defense* policies also are mostly deterministic and consist of hardening the critical node N_{data}, and patching identified vulnerabilities across all nodes. Finally, we also find that the defender learns to use the predictable attack pattern to its advantage and defends where the attacker is likely to attack next.
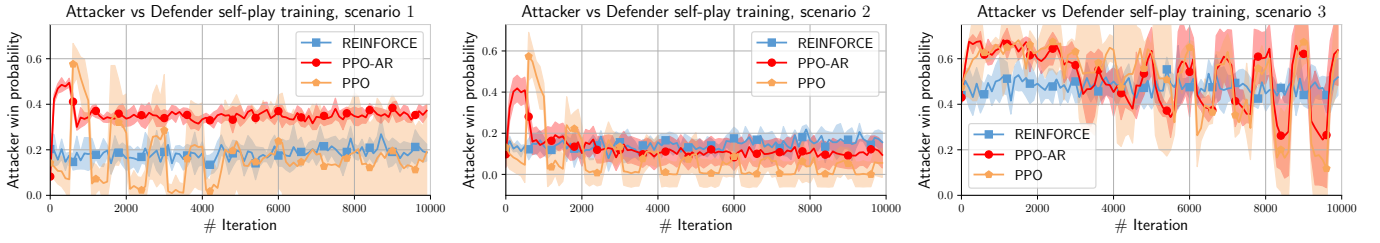
Fig. 5: Attacker win ratio against the number of training iterations; the three sub-graphs show the results from training the attacker and the defender in self-play for the three scenarios; the curve labeled PPO-AR shows the mean values of our proposed method; the results are averages over five training runs with different random seeds; the shaded regions show the standard deviation.

B. Learning Policies Against a Dynamic Opponent

In this subsection we study the setting where the attacker and the defender learn simultaneously. Our goal is to investigate whether this setting can lead to stable and effective policies.

1) Simulation Setup: To evaluate our method, we use the same setup as in Section V-A and run $10,000$ training iterations, each with five different random seeds. After each iteration, *both* agents update their policies. Hence, this setup is different from Section V-A, where one agent is static and only one agent updates its policy after each iteration. That both agents update their policies means that, in each iteration, an agent plays against a different opponent policy, making the environment non-stationary. Consequently, the main challenge in this setting is to get the agents' policies to converge. To improve convergence, we use the opponent pool technique described in Section IV-E. The results are shown in Fig. 5.

2) Analysis of the Results in Fig. 5: We observe in Fig. 5 that policies sometimes converge and sometimes oscillate across iterations. We also see that this behavior is dependent on the particular scenario. For instance, we find that scenario 2, which represents an infrastructure with good defenses (see Section III-C), exhibits stable behavior of the policies. In contrast, scenario 3, which captures an infrastructure with weak defenses, exhibits oscillation of the policies.

Looking at our proposed method, PPO-AR, we can see that it stabilizes in scenario 1 and 2 after some $1,000$ iterations. However, in scenario 3 our method oscillates with increasing amplitude. The oscillation indicates that the agent reacts to the changed policy of its opponent and overfits. Similar behavior of self-play has been observed in related works [30], [13]. Although the opponent pool technique has been introduced to mitigate the oscillation, it is clear from the results that this is insufficient in some scenarios.

3) Inspection of Learned Policies: When inspecting the learned policies, we find that they are fully stochastic. This contrasts with the policies inspected in Section V-A, that we found to be deterministic. By fully stochastic we mean that in a specific state of the game, the policy suggests several actions, each with significant probability. This result indicates that although deterministic policies are effective against static

opponents, they are ineffective when facing adaptive opponents that can exploit the deterministic policy [10], [29]. For example, an adaptive attacker can learn to circumvent a deterministic defense policy. Finally, apart from being stochastic, the converged policies are similar to strategies of humans. In particular, we find that our method discovered stochastic defense policies that combine threat identification with attack forecasting and stochastic attack policies that combine reconnaissance with targeted exploits.

C. Discussion of the Results

The results demonstrate that, for a simple infrastructure, our method outperforms two baselines and discovers effective strategies for intrusion prevention without prior knowledge. When one agent is static in self-play, convergence is stable (Fig. 3) and the agents converge to deterministic policies that are effective against the static opponents. Conversely, when both agents learn simultaneously in self-play, convergence depends on the scenario (Fig. 5) and the agents converge to stochastic policies that are more general.

When inspecting the learned policies, we find that the policies reflect common-sense knowledge and are similar to strategies of humans. Most notably, these policies were discovered using no prior knowledge. This indicates that self-play works as a defense hardening technique. As the defender improves its defense in self-play, the attacker learns increasingly sophisticated attacks to circumvent the defense, leading to an artificial arms-race where both agents improve their policies over time.

In this paper, we have studied a relatively small and simple infrastructure (Fig. 1) and the application of our method to more complex—and more realistic—infrastructures is left for future work. We expect that by adding more complexity to the model, our method will be able to discover increasingly sophisticated strategies. However, we also expect that training times and computational requirements will increase.

Finally, the results also show that convergence of policies in self-play with two adaptive agents remains a challenge. In our results, we observe that the stability of policy convergence differs between the three scenarios (Fig. 5), indicating that the characteristic of the environment is an important factor for convergence in self-play. We also note that all policies

that did converge in self-play with two adaptive agents were stochastic, which suggests that using methods that can represent stochastic policies is important for policy convergence in self-play.

VI. RELATED WORK

Our research extends prior work on reinforcement learning applied to network security, game-learning programs, and game theoretic modeling of security.

A. Reinforcement Learning in Network Security

The research on reinforcement learning applied to network security is in its infancy. Some early works are [25], [31], [2], [32]. For a literature review of deep reinforcement learning in network security, see [9].

The prior work that most resembles ours are [25], and [32]. Both of them experiment with abstract models for network intrusion and defense. Our work can be seen as a direct extension of these works as we use a similar model for our study. Our work in this paper differs in the following ways: (1) we propose an extension of the models in [25], [32] to include reconnaissance; (2) we analyze learned security strategies in self-play; (3) we propose a reinforcement learning method that uses function approximation, an opponent pool, and an autoregressive policy representation; and (4), we extend prior work to consider state-of-the-art algorithms (e.g. PPO [21]).

B. Game-Learning Programs

Games have been studied since the inception of artificial intelligence and machine learning. An early example is Samuel's Checkers player [33]. Other notable game-learning programs are Deep Blue [34] and TD-Gammon [26]. Recent developments include AlphaGo [12], OpenAI Five [13], DeepStack [35], and Libratus [36].

Although we take inspiration from these works, our study differs by focusing on security strategies as opposed to pure game strategies. Moreover, even though our study focuses on self-play training, we also take inspiration from prior work that have investigated learning theory in games, in particular Minimax-Q [23], Nash-Q [37], Neural Fictitious Self-Play [29], and the seminal book by Fudenberg and Levine [38].

C. Game Theoretic Modeling in Cyber Security

Several examples of game theoretic security models can be found in the literature [5], [39], [4]. For example, FlipIt [39] is a security game that models Advanced Persistent Threats (APTs). Other notable works include the book by Alpcan and Basar [4] and the survey by Manshaei et al. [5]. We consider the related work on game theoretic methods to be complementary to the reinforcement learning methods studied in this paper.

VII. CONCLUSION AND FUTURE WORK

We have proposed a method to automatically find security strategies for the use case of intrusion prevention. The method consists of modeling the interaction between an attacker and a defender as a Markov game and letting attack and defense strategies evolve through reinforcement learning and self-play without human intervention. We have also addressed known challenges of reinforcement learning in this domain and presented an approach that uses function approximation, an opponent pool, and an autoregressive policy representation.

Using a simple infrastructure configuration, we have demonstrated that effective security strategies can emerge from self-play. Inspection of the converged policies show that the emerged policies reflect common-sense knowledge and are similar to strategies of humans (e.g. Fig. 4). Moreover, through evaluations, we have demonstrated that our method is superior to two baseline methods (Fig. 3), but that the problem of oscillating policies in self-play remains a challenge (Fig. 5). In particular, we have found a scenario where our method does not lead to converging but oscillating policies.

In future work we plan to further study the observed policy oscillations and explore techniques for mitigation. We also plan to increase the scale of the intrusion prevention use case and model more realistic infrastructures, e.g. a campus network. Finally, we will consider two extensions to our model. First, we plan to enrich the model's action space to cover additional use-cases, e.g. Advanced Persistent Threats (APTs). Second, we plan to lower the level of abstraction of the model to examine if our method can discover practically useful security strategies, i.e. strategies related to specific network protocols and software.

VIII. ACKNOWLEDGMENTS

APPENDIX

A. Hyperparameters

Parameter	Value
Discount factor γ	0.999
Learning rate α	0.0001
# Shared layers	2
# Hidden neurons	128
# Layers for value head	1
# Layers for policy head	1
Batch size	2000
Entropy coefficient	0.001
GAE λ	0.95
Clipping range ϵ	0.2
Optimizer	Adam [40]
Opponent pool max size	100000
Opponent pool increment iterations	50
Opponent pool sample p	0.5

TABLE 1: Hyperparameters.

REFERENCES

[1] M. A. Maloof, *Machine Learning and Data Mining for Computer Security: Methods and Applications (Advanced Information and Knowledge Processing)*. Berlin, Heidelberg: Springer-Verlag, 2005.

[2] K. Durkota, V. Lisy, B. Bošansky, and C. Kiekintveld, "Optimal network security hardening using attack graph games," in *Proceedings of the 24th International Conference on Artificial Intelligence*, ser. IJCAI'15. AAAI Press, 2015, pp. 526–532. [Online]. Available: http://dl.acm.org/citation.cfm?id=2832249.2832322

[3] P. Johnson, R. Lagerström, and M. Ekstedt, "A meta language for threat modeling and attack simulations," in *Proceedings of the 13th International Conference on Availability, Reliability and Security*, ser. ARES 2018. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3230833.3232799

[4] T. Alpcan and T. Basar, *Network Security: A Decision and Game-Theoretic Approach*, 1st ed. USA: Cambridge University Press, 2010.

[5] M. H. Manshaei, Q. Zhu, T. Alpcan, T. Basar, and J.-P. Hubaux, "Game theory meets network security and privacy," *ACM Comput. Surv.*, vol. 45, no. 3, pp. 25:1–25:39, Jul. 2013. [Online]. Available: http://doi.acm.org/10.1145/2480741.2480742

[6] R. Bronfman-Nadas, N. Zincir-Heywood, and J. T. Jacobs, "An artificial arms race: Could it improve mobile malware detectors?" in *2018 Network Traffic Measurement and Analysis Conference (TMA)*, 2018, pp. 1–8.

[7] S. Noreen, S. Murtaza, M. Z. Shafiq, and M. Farooq, "Evolvable malware," in *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 1569–1576. [Online]. Available: https://doi.org/10.1145/1569901.1570111

[8] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, ser. SP '10. USA: IEEE Computer Society, 2010, p. 305–316. [Online]. Available: https://doi.org/10.1109/SP.2010.25

[9] T. T. Nguyen and V. J. Reddi, "Deep reinforcement learning for cyber security," *CoRR*, vol. abs/1906.05799, 2019. [Online]. Available: http://arxiv.org/abs/1906.05799

[10] M. Bowling and M. Veloso, "Scalable learning in stochastic games," in *AAAI Workshop on Game Theoretic and Decision Theoretic Agents*, July 2002.

[11] S. Li, Y. Wu, X. Cui, H. Dong, F. Fang, and S. Russell, "Robust multi-agent reinforcement learning via minimax deep deterministic policy gradient," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 4213–4220, 07 2019.

[12] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016.

[13] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. W. Pachocki, M. Petrov, H. P. de Oliveira Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang, "Dota 2 with large scale deep reinforcement learning," *ArXiv*, vol. abs/1912.06680, 2019.

[14] O. Vinyals, I. Babuschkin, W. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. Agapiou, M. Jaderberg, and D. Silver, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, 11 2019.

[15] R. Bellman, "A markovian decision process," *Journal of Mathematics and Mechanics*, vol. 6, no. 5, pp. 679–684, 1957. [Online]. Available: http://www.jstor.org/stable/24900506

[16] R. A. Howard, *Dynamic Programming and Markov Processes*. Cambridge, MA: MIT Press, 1960.

[17] W. S. Lovejoy, "A survey of algorithmic methods for partially observed markov decision processes," *Annals of Operations Research*, vol. 28, no. 1, pp. 47–65, Dec 1991. [Online]. Available: https://doi.org/10.1007/BF02055574

[18] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[19] R. Bellman, *Dynamic Programming*. Dover Publications, 1957.

[20] C. Watkins, "Learning from delayed rewards," Ph.D. dissertation, 01 1989.

[21] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[22] L. S. Shapley, "Stochastic games," *Proceedings of the National Academy of Sciences*, vol. 39, no. 10, pp. 1095–1100, 1953. [Online]. Available: https://www.pnas.org/content/39/10/1095

[23] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Proceedings of the Eleventh International Conference on International Conference on Machine Learning*, ser. ICML'94. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1994, p. 157–163.

[24] S. Li, Y. Wu, X. Cui, H. Dong, F. Fang, and S. Russell, "Robust multi-agent reinforcement learning via minimax deep deterministic policy gradient," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 4213–4220, 07 2019.

[25] R. Elderman, L. J. J. Pater, A. S. Thie, M. M. Drugan, and M. Wiering, "Adversarial reinforcement learning in a cyber security simulation," in *ICAART*, 2017.

[26] G. Tesauro, "Td-gammon, a self-teaching backgammon program, achieves master-level play," *Neural Comput.*, vol. 6, no. 2, p. 215–219, Mar. 1994. [Online]. Available: https://doi.org/10.1162/neco.1994.6.2.215

[27] R. J. Williams, "Reinforcement-learning connectionist systems," 1987.

[28] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[29] J. Heinrich and D. Silver, "Deep reinforcement learning from self-play in imperfect-information games," *CoRR*, vol. abs/1603.01121, 2016. [Online]. Available: http://arxiv.org/abs/1603.01121

[30] D. Hernandez, K. Denamganaï, Y. Gao, P. York, S. Devlin, S. Samothrakis, and J. A. Walker, "A generalized framework for self-play training," in *2019 IEEE Conference on Games (CoG)*, 2019, pp. 1–8.

[31] K. Chung, C. A. Kamhoua, K. A. Kwiat, Z. T. Kalbarczyk, and R. K. Iyer, "Game theory with learning for cyber security monitoring," in *Proceedings of the 2016 IEEE 17th International Symposium on High Assurance Systems Engineering (HASE)*, ser. HASE '16. Washington, DC, USA: IEEE Computer Society, 2016, pp. 1–8. [Online]. Available: http://dx.doi.org/10.1109/HASE.2016.48

[32] A. Ridley, "Machine learning for autonomous cyber defense," 2018, the Next Wave, Vol 22, No.1 2018.

[33] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM J. Res. Dev.*, vol. 3, no. 3, p. 210–229, Jul. 1959. [Online]. Available: https://doi.org/10.1147/rd.33.0210

[34] M. Campbell, A. J. Hoane, and F.-h. Hsu, "Deep blue," *Artif. Intell.*, vol. 134, no. 1–2, p. 57–83, Jan. 2002. [Online]. Available: https://doi.org/10.1016/S0004-3702(01)00129-1

[35] M. Moravčík, M. Schmid, N. Burch, V. Lisý, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, and M. Bowling, "Deepstack: Expert-level artificial intelligence in heads-up no-limit poker," *Science*, vol. 356, no. 6337, pp. 508–513, 2017. [Online]. Available: https://science.sciencemag.org/content/356/6337/508

[36] N. Brown and T. Sandholm, "Superhuman ai for heads-up no-limit poker: Libratus beats top professionals," *Science*, vol. 359, no. 6374, pp. 418–424, 2018. [Online]. Available: https://science.sciencemag.org/content/359/6374/418

[37] J. Hu and M. P. Wellman, "Nash q-learning for general-sum stochastic games," *J. Mach. Learn. Res.*, vol. 4, no. null, p. 1039–1069, Dec. 2003.

[38] D. Fudenberg and D. Levine, *The Theory of Learning in Games*, 1st ed. The MIT Press, 1998, vol. 1. [Online]. Available: https://EconPapers.repec.org/RePEc:mtp:titles:0262061945

[39] M. van Dijk, A. Juels, A. Oprea, and R. L. Rivest, "Flipit: The game of "stealthy takeover"," *Journal of Cryptology*, vol. 26, no. 4, pp. 655–713, Oct 2013. [Online]. Available: https://doi.org/10.1007/s00145-012-9134-5

[40] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015. [Online]. Available: http://arxiv.org/abs/1412.6980