1.5em 0pt

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2023.0322000

Scalable and Autonomous Network Defense using Reinforcement Learning

ROBERT G. CAMPBELL[1], MAGDALINI EIRINAKI[2] (Member, IEEE), and YOUNGHEE PARK[3] (Member, IEEE)

[1]San Jose State University, San Jose, CA 95192 USA (e-mail: robertgcampbell95@gmail.com)
[2]San Jose State University, San Jose, CA 95192 USA (e-mail: magdalini.eirinaki@sjsu.edu)
[3]San Jose State University, San Jose, CA 95192 USA (e-mail: younghee.park@sjsu.edu)

Corresponding author: Magdalini Eirinaki (e-mail: magdalini.eirinaki@sjsu.edu)

ABSTRACT An autonomous network defense method under attack is a critical part of preventing network infrastructure from potential damage in real time. Despite various network intrusion detection techniques, our network space is not safe enough due to the increasing exploitation of software vulnerabilities. Thus, timely response and defense methods under network intrusion are important techniques given the large scope of cyberattacks in recent years. In this paper, we design a scalable and autonomous network defense method by using the model of a zero-sum Markov game between an attacker and a defender agent. To scale up the proposed defense model, we utilize a graph convolutional network (GCN) along with framestacking to address the partial observability of the environment. The agents are trained using Proximal Policy Optimization (PPO) which allows for good convergence in a reasonable timeframe. In experiments, we evaluate the proposed model under the large network size while simulating network dynamics including link failures and other network events. The experimental results demonstrate that the proposed method scales well for larger networks and achieves state of the art results on various threat scenarios.

INDEX TERMS Reinforcement learning, network defense, Markov games, deep learning, graph convolutional networks

I. INTRODUCTION

Timely defense against network intrusions is paramount in interconnected digital ecosystems. As we rely extensively on networked systems to facilitate communication and critical operations, the threat landscape has evolved, giving rise to sophisticated and diverse cyber threats with national security implications [1] [2]. This necessitates the development and implementation of robust network defense methods to safeguard against unauthorized access and malicious activities in the network. Thus, recent research has explored the dynamic landscape of network intrusion detection, delving into innovative approaches, advancements in machine learning algorithms, and the integration of artificial intelligence to enhance the efficacy of network defense techniques.

Reinforcement learning (RL) is a rapidly growing sub-field of machine learning that allows an agent to learn from interactions with an environment where the agent receives feedback in the form of rewards [3]. Through these interactions, the agent learns a policy which informs how it interacts with the environment, improving its average reward over time. Agents can also cooperate with or compete in an environment through repeated interactions with other agents and the environment. Cybersecurity, particularly network defense, is a good domain for applying RL agents as the threat landscape is constantly evolving [4] [5]. It is naturally adversarial and multi-agent where adversaries can be humans or automated agents.

In this study we propose a RL approach trained using proximal policy optimization (PPO) to train an agent to choose countermeasures for defending a network. The goal of the defender is to prevent the compromise of a critical node while minimizing the impact of the attacker by the network. The agents compete in a zero-sum game with rewards for each agent based on the outcome of the game. We model both the defender and the attacker as RL agents with a tournament-based training regime. We leverage our previous work and use a training curriculum which incorporates various network topologies to train more general agents [6]. The agents are evaluated in varying environments and with varying simulated threat scenarios in order to determine the effect of observed vulnerabilities in the environment on defender performance.

Prior methods test their approach on a single topology

or on a fixed network size [5] [7] [8] [9]. However, real world networks are rarely fixed and often add or remove hosts based on demand, link failures, outages, or other factors. Contrary to previous work that trains on a fixed topology and network size, we instead evaluate our method across threat scenarios with varying network sizes. We propose a scalable approach that uses a new reward function and a graph convolutional network (GCN) along with framestacking to address the partial observability of the environment [10]. We choose Proximal Policy Optimization (PPO) for training the agents, which offers a good balance of computational complexity and convergence speed. As shown in the experimental evaluation, the proposed reward function achieves higher defender win rates compared to the previously proposed reward [6] [7], whereas the GCN architecture enables the model to scale to larger network sizes, while also outperforming the MLP architecture [6] across all network sizes, demonstrating the capabilities of the proposed approach.

The remainder of this paper is organized as follows. A review of the state-of-the-art is given in Section II. A breakdown of the environment, state, and actions of the Markov game is given in Section III. This section also includes a discussion of the proposed rewards and observations per agent as well a discussion of self-play and the proposed model architecture. In Section V, we present the results of our experimental evaluation across various parameters, including framestacking, reward, threat scenarios, and network size. Finally, in Section VI, we discuss the results and give concluding remarks as well as future research directions of the work.

II. RELATED WORK

We start by providing some necessary background and establishing the terminology on Reinforcement Learning (RL) and Multi-Agent Reinforcement Learning (MARL). We then proceed to discuss in more detail the state-of-the-art approaches that have employed RL/MARL for network defense.

A. REINFORCEMENT LEARNING

Reinforcement learning is the process of learning through interactions between the agent and the environment with reward as feedback. The interaction between an agent and its environment is represented by a Markov Decision Process (MDP). This model consists of pairs of states and actions, denoted as (s_t, a_t), which helps in determining the likelihood of transitioning between states s in the environment following an action a at a specific time t. An enhancement of this model, known as the Markov Reward Process (MRP), incorporates rewards R for each state-action pair (s_t, a_t) [11]. The key feature of Markov processes is their lack of memory, meaning that only the current state matters, rendering historical data irrelevant [12] [13]. A Partially Observable Markov Decision Process (POMDP) differs from an MDP in that it includes states $s \in S$ which are not fully visible to the agent. Akin to MRP, rewards can also be integrated into POMDP, forming a Partially Observable Markov Reward Process (POMRP) [14]. POMDPs are characterized by an observation function $s_t \rightarrow O_t$, which maps states s to observations O at time t [12]. Markov decision processes can be expanded to include multiple agents, from one to any number $n >= 2$, and are termed Markov games. These games focus on identifying a set of policies that form a Nash equilibrium among competing agents, rather than solely aiming to maximize the sum of the expected rewards. However, the learned Nash equilibrium can be sub-optimal [13].

RL algorithms fall into two major categories: policy-based and value-based methods. Value based networks such as Deep Q Networks (DQN) learn to predict the expected total reward for a given state s when action a is performed [15]. The action with the highest Q value is then the action with the highest expected return. In contrast, policy based methods directly optimize for the policy π. Approaches such as PPO are policy based but augment with a value based critic to minimize variance in policy updates [16].

B. MULTI-AGENT REINFORCEMENT LEARNING

Multi-agent reinforcement learning extends the ideas in single-agent RL and applies them to Markov games. Markov games can be either cooperative or competitive. A well known issue of MARL is the non-stationarity of the environment, which violates the Markov property. This refers to fact that from each agent's perspective, the actions of additional agents form part of the environment. Consequently, the moving policy distribution of the agents inhibit convergence [17].

Markov games where the possible actions and optimal strategies are the same for all participants are known as symmetric games. In 2017, Silver et al. used train an agent to achieve grandmaster level play in Go [18]. The authors used Monte Carlo Tree Search (MCTS) to simulate games. Owing to the symmetric nature of the game, the authors were able to use self-play to train the agent against itself and thus learns to play Go from zero initial knowledge.

In asymmetrical games, the players have different sets of actions and objectives. In 2019, OpenAI trained groups of collaborating agents to compete in a simulated game of hide-n-seek. The multi-agent competition creates a natural auto-curricula where agents are always playing againts adversaries with a similar skill level [19].

Most recently, Schrittwieser et al. expanded on the search based framework built by AlphaZero in a work titled MuZero [20]. MuZero is a model-based approach that outputs three values at each step: actions, values, and rewards. The model uses a recurrent model internally that takes the observation and possible future action as input. The authors tested MuZero using self-play in various environments such as chess, Go, and Atari.

C. ML FOR NETWORK DEFENSE

Machine learning has been extensively used in cybersecurity research, particularly in network intrusion systems. Wang et al. proposed a system for learning feature representations to improve classification accuracy. This approach uses an autoencoder to encode categorical features which when con-

catenated to the numerical features is then encoded for use by the downstream classifier [21].

Reinforcement learning can extend this paradigm by not just identifying attacks but also responding in real-time. For example, attackers can compromise devices and hide illicit activity as spikes in traffic which can lead to packet loss. Dake et al. [22] propose a RL system to detect DDOS and other suspicious activity in an IOT network. The authors highlighted three types of attacks in the network: traffic burst, elephant flow, and DDOS in the network. An RL agent is then trained to address to manage these events by installing flow rules through the software-defined networks (SDN) controller to manage traffic.

Reinforcement Learning (RL) has been utilized as a method for intrusion prevention. In this context, an intrusion prevention system, integrated within the controller of SDNs, leverages feedback from an intrusion detection system to slow down or avert cyber attacks. Han et al. [8] explored the use of a defense agent and the potential adversarial threats to RL agents within a centralized network controller. Similarly, Gabirondo-Lopez et al. [7] investigated the application of RL in training an intrusion prevention system. Their approach frames the training as a simultaneous competition between an attacking and a defending agent, essentially a zero-sum Markov game. In their model, the network is depicted as a fully interconnected system with a distinct, segregated honeynet, allowing the defender to quarantine compromised hosts.

In this study, we also modeled the problem as a zero-sum Markov game between an attacker and a defender. Previous methods evaluated their approach on a single topology or limited the agents to a subset of the network [7] [4] [8] [5]. Instead, we leverage our previous work which proposed a dynamic topology curriculum, enabling the training of more general, capable agents [6]. In that study, we evaluated various threat scenarios in terms of exploitability and impact and concluded that the curriculum improves the defender's win rate over training on a static topology by exposing the agent to more challenging environments over time. While our previous work showed promising results, the framework architecture, based on a multi-layer perceptron (MLP)-based network for the value and policy models, was not scalable, whereas the reward function used, which was that of Gabirondo-Lopez et al. [7], worked well with small networks but performed poorly as we scaled the environment.

III. MULTI-AGENT REINFORCEMENT LEARNING FOR AUTONOMOUS NETWORK DEFENSE

The agents engage in competition within a simulated network setting, as modified by Campbell et al. [6], [23] from the original concept by Gabirondo et al. [7]. In this environment, the attacker's goal is to infiltrate the critical node and capture as many hosts within the network as possible. On the other hand, the defender aims to protect the critical node and minimize the attacker's influence on the network. The competition concludes either when one agent emerges victorious or when the predefined duration of the episode is met, in which case it is considered a tie. The agents take turns to act, starting with the attacker and followed by the defender. Following each action, the conditions for winning are evaluated, and at the end of the game, rewards are allotted based on these results.

In the following sections, we describe the environment, review the observations and actions [6], [23], and define the reward function of the agents.

TABLE 1. CVSS Vulnerability Severity Ratings

Rating	Base Score Range
None	0
Low	0.1 - 3.9
Medium	4.0 - 6.9
High	7.0 - 8.9
Critical	9.0 - 10.0

A. ENVIRONMENT

The environment is represented as a $k \times k$ graph G, where k is the number of nodes in the network. Similar to an adjacency matrix, G encodes the neighbors of the nodes on the off-diagonal. Additionally, the value on the diagonal $h_i \in G$ represents the node state, where $i \in [0, k-1]$.

Each agent has a corresponding observation of the environment as the environment is partially observable from the perspective of both the attacker and the defender. However, the defender has more information of the environment's true state, in this case the links between nodes, since we assume the defender is interacting with the network through the SDN controller.

1) Vulnerability Scores

Each node has a vulnerability that can be exploited by the attacker. These vulnerabilities are modeled using the NIST Common Vulnerability Scoring System (CVSS) [24]. The CVSS is a framework used by the National Vulnerability Database (NVD) to categorize and rank vulnerabilities. The scoring system rates vulnerabilities as *none*, *low*, *medium*, *high* or *critical* risk using a vulnerability score with range 1-10 as shown in Table 1. This score, also known as the base score, is what we use in lieu of early warning alerts from an intrusion detection system (IDS). The components of the base score are used to determine the success of certain actions in the environment. This idea was introduced in [7] and we include it here for completeness.

The base score is composed of two subscores: the *Impact* score and the *Exploitability* score [24].

The equations for impact score is defined as:

$$IScore = 1 - (1 - C)(1 - I)(1 - A) \tag{1}$$

where C is the confidentiality, I is the integrity, and A is the availability. The components measure the impact to confidentiality, integrity, and availability as a result of the exploited vulnerability. The exploitability score is defined as:

$$EScore = 8.22 \times AV \times AC \times PR \times UI \tag{2}$$

where AV represents the attack vector, AC the attack complexity, PR the required privilege, and UI the user interaction to carry out the exploit.

With the impact and exploitability, the base score is then defined as:

$$BaseScore = \begin{cases} 0 & IScore = 0, \\ round(\min((IScore + EScore), 10)) & \text{otherwise} \end{cases} \quad (3)$$

As part of the global state there exists a mapping $v_i \rightarrow VulnScores$, where each entry is a tuple $(IScore, EScore, BaseScore)$.

The scores are utilized to evaluate the effectiveness of specific actions by both agents. The IScore and EScore are derived from a Gaussian distribution for each network node, contributing to a training curriculum. This approach allows for incrementally increasing the standard deviation of the parameters, thereby challenging the agents with increasingly complex environments. These range from low severity scenarios (characterized by low exploitability and impact) to critical ones (with high exploitability and impact), as outlined in Table 1. The parameters of the distribution are adjusted to simulate various threat situations.

This work presumes that an intrusion detection system provides the scores. In practical applications, these vulnerability scores can be integrated with other system monitoring the network [25].

2) Topology

In order to test the approach against a variety of network environments, we use the proposed training curriculum framework found in [6]. The advantages of that approach is topologies are varied over time, with the defender first exposed to simple linear networks and encountering networks with more edges between nodes as training progresses. The curriculum also uses a fixed starting value of $IScore$ and $EScore$ that increases in variance as training progresses. These features therefore expose the defender agent to increasingly more challenging environments, from both a topological and security perspective.

B. OBSERVATIONS

As previously stated, both agents only partially observe the environmental state. There is a global state that accurately reflects the environment's condition. Each episode starts with the agents' starting nodes being randomized. The attacker's starting point involves one node being designated as compromised, with the observation updated accordingly. Meanwhile, the defender's starting node is identified as the critical (flag) node. This approach of randomizing start positions is employed to avoid the agents becoming overly accustomed to specific hosts (nodes). Figure 1 provides a key for understanding the states of the nodes.

At each step, we update the observation for the respective agent given action a_t. First we decompose the action into the target node and target action. For both agents, we validate the action given the observation and the action subcomponents

FIGURE 1. Color key and state value for node states.

$a_{t,n}, a_{t,a}$ where $a_{t,n}$ is the component of the action at time t targeting node n and $a_{t,a}$ is the component of the action at time t performing action a on the targeted node. If the action is valid, then the action is performed (step).

Additionally for the attacker, we find a subset of nodes $N \subseteq V$, where V is the set of nodes in the network. N is the set of nodes that are one hop away from a compromised node whose vulnerabilities are not exploited (state 0 or 1). If either N is not empty or the target node is compromised then we take the action on the selected target node.

C. REWARDS

The reward is comprised of two subcomponents: the ratio of the nodes compromised in the network and the ratio of steps remaining in the episode to episode horizon. The reward for the attacker and the defender is proportional and inversely proportional to the number of compromised nodes respectively. Both attacker and defender are rewarded for winning the game in the least number of steps possible. The reward R_w is defined in Equation 4.

$$R_w = \begin{cases} \frac{CN}{TN} + \frac{SR}{ST} & \text{winner = attacker}, \\ 1 - \frac{CN}{TN} + \frac{SR}{ST} & \text{winner = defender}, \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

where CN refers to the number of compromised nodes and TN refers to the total number of nodes in the network. The second term in the equation accounts for how quickly the episode ends, with SR referring to the number of steps remaining in the episode and ST referring to the total number of steps (horizon).

D. ACTIONS

Similar to previous work ([6], [7]) we simplify the action space to three actions per agent. This allows us to use the same neural network architecture for both agents while training both policies separately. An overview of the actions is given in Table 2.

1) Attacker Actions

The attacker's objective is to compromise the critical node as shown in Figure 2. As such, the agent's actions are directed at exploring the network and identifying vulnerabilities in order to exploit them. This model of the attacker is based on a scenario where an attacker has gained access to a private network through a publicly available service or host. Each episode, the attacker starts with a single randomly selected compromised node. From there, the attacker can explore and exploit to compromise the critical node. The attacker Figure

TABLE 2. Agent action overview

Agent	Action	Description
Attacker	Explore Topology	Update observation of compromised node
	Scan Vulnerability	Scan host for vulnerabilities
	Exploit vulnerability	Exploit vulnerability on host
Defender	Check Status	Update observation of target node
	Isolate Node	Isolate compromised node
	Migrate Node	Migrate critical node

3 shows an example of the actions in the environment in the clique topology as well as the corresponding attacker observations. Algorithm 1 details the attacker's action step.

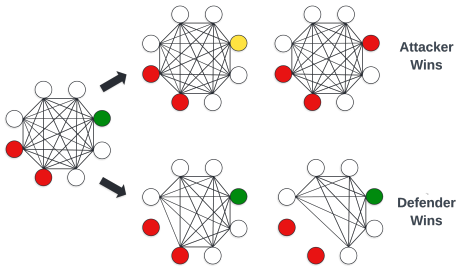

FIGURE 2. Win conditions example.

Algorithm 1 Attacker Agent Step

procedure AttackerStep($O_a, a_{t,n}, a_{t,a}$)
  $exploitabilityRatio = O_g[exploitabilityScore][a_{t,n}]/10$
  if $a_{t,a} = 0$ **then** ▷ Explore Topology
    $setNeighbors(O_a, a_{t,n}, O_g)$
  end if
  if $a_{t,a} >= 0$ **then** ▷ Scan or Exploit Vulnerability
    $actionSuccess = random()$
    if $actionSuccess < exploitabilityRatio$ **then**
      $O_a[a_{t,n}][a_{t,n}] \leftarrow a_{t,a}$
      $O_g[a_{t,n}][a_{t,n}] \leftarrow a_{t,a}$
    end if
  end if
end procedure

Explore Topology: This action updates the attacker's state based on the global state. This action incorporates the neighboring nodes of the chosen target into the attacker's observation. This action is limited to nodes that the attacker has compromised; thus, any attempt to apply it to nodes in different states will be unsuccessful. Additionally, if the defender has succeeded in isolating the node, this particular action will be rendered ineffective.

Scan Vulnerability: Performs a scan of the target node for vulnerabilities. The success of this action depends on the *EScore* of the targeted node. Target node must be in the normal state.

Exploit Vulnerability: Exploits the scanned vulnerability. Target node must be already scanned for the action to succeed. If the critical node is exploited by the attacker, the attacker wins and the game ends.

2) Defender Actions

The objective of the defender was to protect the critical node while minimizing impact to the network and its services. Impact is measured by the number of nodes compromised by the attacker and whether the critical node has been compromised. Figure 2 shows the potential win conditions for both agents. The defender's actions allow it to update its observation and isolate known compromised nodes. The defender may also migrate the critical node to another location in the network. Figure 4 shows an example of the actions and the defender observations in the environment. Algorithm 2 demonstrates the logic behind the defender's action in greater detail.

FIGURE 3. Example of attacker actions.

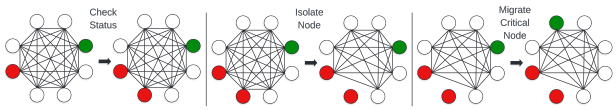

FIGURE 4. Example of defender actions in clique topology.

Check Node Status: If a node is compromised, the likelihood of discovering the exploited vulnerability is:

$$P[O_d(h_i) = 2 | O_g(h_i) = 2] = \frac{BaseScore}{10log_2(k)} \quad (5)$$

where k is the number of nodes in the network, O_d is the defender's observation, O_g is the global state, and h_i is the index of the host (node) [6].

Isolate Node: The defender can mitigate the impact of the attacker in the network by isolating the node once an exploited vulnerability has been discovered in the network. This is a deterministic action that removes the neighbors from the targeted host and updates the global state.

Migrate Node: This actions provides the defender with the option to migrate the critical node to a different node within the network. Initially, the defender verifies whether its observation aligns with the observation in the overall network state. This verification is identical to the one conducted in the *Check Node Status* action. In cases where the global state indicates that the target host is compromised and this verification is successful, the defender then revises its observation.

Algorithm 2 Defender Agent Step

procedure DefenderStep($O_a, a_{t,n}, a_{t,a}$)
  $baseScore \leftarrow O_g[baseScore][a_{t,n}]$
  $targetNodeState \leftarrow O_g[networkGraph][a_{t,n}]$
  $actionSuccessThreshold \leftarrow \frac{baseScore}{10 \times \log_2(k)}$
  $actionSuccess = random()$
  if $a_{t,a} = 0$ **then**
    if $targetNodeState = 0$ **then** ▷ Check status
      if $actionSuccess < actionSuccessThreshold$ **then**
        $O_a[a_{t,n}][a_{t,n}] \leftarrow 2$
        $addCountermeasureCost(a_{t,a})$
      end if
    end if
  end if
  if $a_{t,a} = 1$ **then** ▷ Isolate compromised node
    $setNeighbors(O_a, a_{t,n}, 0)$ ▷ Set edges for node to zero
    $addCountermeasureCost(a_{t,a})$
  end if
  if $a_{t,a} = 2$ **then** ▷ Migrate critical node
    if $targetNodeState = 2$ **then**
      if $actionSuccess < actionSuccessThreshold$ **then**
        $O_a[a_{t,n}][a_{t,n}] \leftarrow 2$
        $addCountermeasureCost(a_{t,a})$
      end if
    end if
    if $targetNodeState \neq 2$ **then**
      $criticalNode \leftarrow indexOf(O_a, 3)$
      $O_a[criticalNode][criticalNode] \leftarrow 0$
      $O_a[a_{t,n}][a_{t,n}] \leftarrow 3$
    end if
  end if
end procedure

Conversely, if the global state suggests that the target host is not compromised, the node is then migrated. It's important to note that due to the probabilistic nature of the *Check Node Status* action, there's a possibility that the defender fails to detect the compromised node.

IV. TRAINING PROCESS

A. SELF-PLAY

Similar to previous work [6], [26], we use self-play to train the agents. Rather than facing the latest adversary, we create and manage a pool of past opponents for each agent. Subsequently, the attacker and defender alternate in competing against adversaries from their own respective pools. Additionally, it's important to differentiate between agents and policies in this scenario. There are two main trainable policies: the attacker and the defender. Each agent maps to many policies, which are snapshots at different points during the training process.

The trainable policies only play against the snapshot policies in the opponent pools. Before training, opponent pools were initialized with a random policy for each respective opponent pool. At each training iteration, the weights of the past opponent were averaged with the weights of the recently trained policy for that agent. This new policy is then added to the opponent pool.

1) Opponent Sampling

Instead of a uniform sample of opponents [26], we considered a split opponent pool with a bias towards more recent agents similar to the approach in [17]. We define three quantities: (a) the size of the recent agent pool, expressed as a percentage d, (b) the probability p_{recent} of sampling from the recent agent pool, and (c) the probability of sampling one of the $(1-d)\%$ agents, $p_{past} = 1 - p_{recent}$. Over training, we decayed the value of p_{recent} to gradually increase the chance of training against previous policies. This was done to address the issue of catastrophic forgetting.

B. MODEL ARCHITECTURE

In our previous work we used a multi-layer perceptron (MLP) based network for the value and the policy models [6]. The main problem with using an MLP is scaling, as the model parameter count increases exponentially as the number of nodes increases. To better scale our method, we propose a policy/value model that uses a graph convolutional network (GCN) for feature extraction using a graph-level embedding. A comparison between the number of parameters for the MLP and the proposed GCN is shown in Figure 5. The figure shows that the number of parameters increases exponentially for the MLP whereas the GCN parameter count increases linearly. This is due to two architectural details. The GCN weights are shared across the frames of the input observations; additionally, the number of units of the GCN remain fixed regardless of the network size.

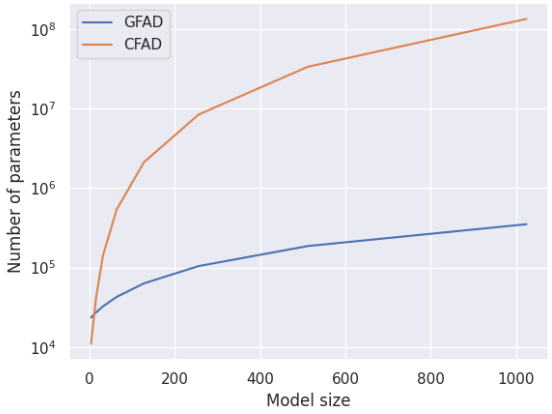

FIGURE 5. Comparison of model parameter count between GFAD and CFAD. Y-axis uses log scaling.

In order to address the partial observability of the environment, the model takes as input a stack of the the last m observations [10]. As shown in Figure 6, this stack of observations are preprocessed further so that the node states are extracted from the observation as features for each node. The edges and the node features were passed to the graph convolutional layers, with the GCN applied to each observation in the framestack. This produces graph embeddings per each observation which are then concatenated. The node embeddings are aggregated using a self attention global pooling layer which selects the most relevant node embeddings [27].

In addition to the observations the model also takes as input the past n actions as integer labels. This stack of actions is one-hot encoded and concatenated into a single vector which

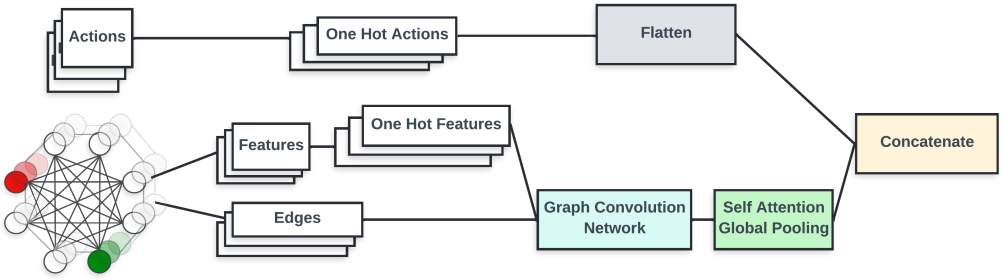

FIGURE 6. **In depth look at model input preprocessing steps.**

TABLE 3. **Model Architecture Parameters**

Layer	Num units	Number of Layers
GCN	32	1
Global Attention Pooling	32	1
Dense	64	2

TABLE 4. **PPO Parameters**

	Attacker	Defender
Lambda	0.95	0.95
Gamma	0.99	0.99
Clip	0.1	0.1
Value Loss Coefficient	0.3	0.7
Entropy Loss Coefficient	0	0
KL Loss Coefficient	0.8	0.3
Initial Learning Rate	3e-4	3e=4

is then concatenated to the graph embeddings. This representation is then passed to the a stack of dense layers, with a separate stack of layers for the policy output and the value outputs. The overview of the architecture is shown in Figure 7. Layer sizes for the subcomponents of the architecture are detailed in Table 3.

C. TRAINING SETUP

We use the framework Rllib with Ray which simplifies the training loop [28]. This allows us to use the built-in implementation of PPO in Rllib. Ray also allowed us to run environments in parallel allowing sample collection at scale. The environment has a small memory footprint and therefore scales well for training. The configuration details are presented in Table 4. Additional information can be found in [29]. In addition, we use the PettingZoo framework which allows us to use an OpenAI Gym-like API to interact with the multi-agent environment [30], [31]. Tensorflow was used as the deep learning backend and use Stellargraph and Spektral to create the graph convolution portion of the policy and value models [32]. The training utilized 2 CPUs for sample collection along with a GPU for model training with 5 environments.

The learning rate was annealed from 3×10^{-4} to 5×10^{-5} over the course of training. Each training iteration consists of a bactch size of $10,000$ steps, with a batch size of $2,000$ samples for the stochastic gradient descent update. Additional parameters for PPO are shown in Table 4.

V. EXPERIMENTAL EVALUATION

We train agents using the curriculum framework we introduced in [6]. As mentioned previously, real life networks are dynamic; nodes are added to networks, removed from the network, and links can change between nodes. This training curriculum accounts for the networks' dynamic nature. The objective was to demonstrate the effectiveness of the training method for scaling to larger network sizes.

The section begins with a discussion of stacking frames of the observation to address the partial observability of the environment. We tested the approach using no framestack (frame of size one) and compared with frame sizes of two and three to determine which offers the best performance. Secondly, the proposed reward is compared to the reward introduced in [7] and modified in [6]. This comparison is important as prior work demonstrates the current method does not scale well to larger network sizes. To evaluate the scalability of the approach, we trained the dynamic training curriculum for varying network sizes. We use the network of size 16 as a baseline and train with sizes 8, 16, 32, 64, and 128.

We also consider the impact of episode horizon on defender win rate. For this section, the agent is trained with varying horizon sizes and defender win rate is compared across experiments.

Finally, we demonstrate the effectiveness of the approach on varying threat scenarios. These threat scenarios are built by adjusting the *IScore* and *EScore*. Unless otherwise noted, we trained all agents for a total of 5 million timesteps. The default network size is set to 16 nodes. The mean $Impact = 4.31$ and the mean $Exploitability = 2.59$, again unless otherwise noted.

A. FRAMESTACK

We begin by investigating the utility of frame stacking to address the partial observability of the environment. The experiment compared three values for the framestack size $m \in [1, 2, 3]$. The framestack parameter m adjusts the frame size for both the previous observations and actions.

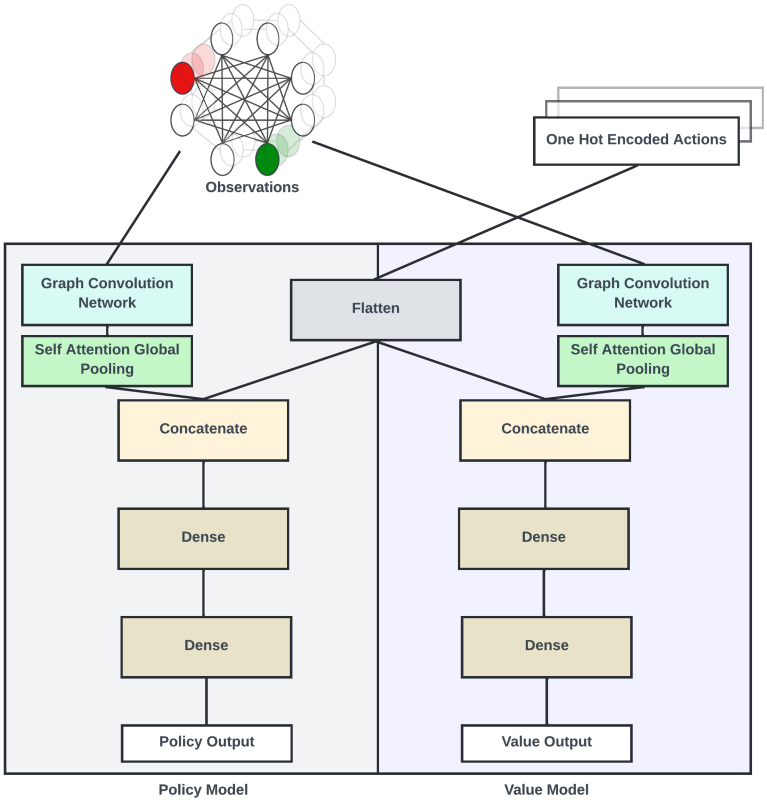

FIGURE 7. **Overview of model architecture.**

There was no significant difference between the win rate and the average reward across the tested values of the framestack size. However, Figure 8 shows that the entropy for the defender's actions decreased with higher values of m. Based on these results, we used a framestack value $m = 2$ for the following experiments to balance memory usage and agent performance.

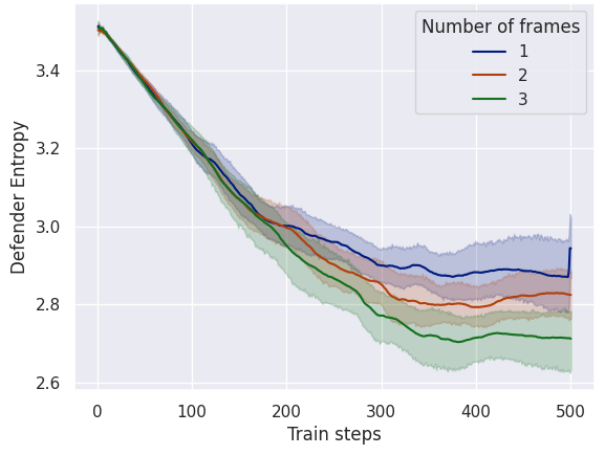

FIGURE 8. **Average defender entropy with confidence interval.**

B. REWARD

The reward function proposed in this work, further referred to as GFAD (for Graph Network Autonomous Defense) is compared to prior work [6], further referred to as CFAD (for Curriculum Framework Autonomous Defense), as shown in Figure 9. This experiment uses the GCN model to compare the reward results using the same model. Since the scale of the rewards is different between the two reward schemes, we instead compare them using the defender win rate.

Figure 9 demonstrates that the defender performs better with the proposed reward. The defender maintains a high win rate throughout training, which suggests that the defender is more robust to changes in the environment and changes in the opponent's policy. Additionally, as shown in Table 5, we observe that that GFAD scales better than CFAD but leave more detailed discussion to Section V-E.

C. EPISODE HORIZON

The episode horizon length is a key aspect of the Markov game as longer episode lengths could more accurately model a real life scenario. To test the defender agent performance, we compare defender win rate across six values of horizon length $h \in [100, 150, 200, 250, 300]$.

As shown in Figure 10 , shorter episode lengths generally lead to lower defender win rate. Once episode length reaches

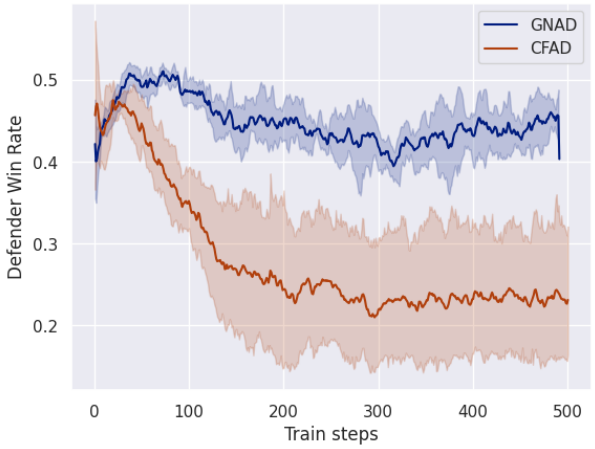

FIGURE 9. Average defender win rate with confidence interval of our proposed reward and the reward proposed in [7] [6].

200 steps, defender win rate plateaus. Since the difference between the results for episode lengths $h \in [200, 250, 300]$ are not statistically significant, the episode length of 200 was chosen as it performs similarly while offering the benefit of faster convergence.

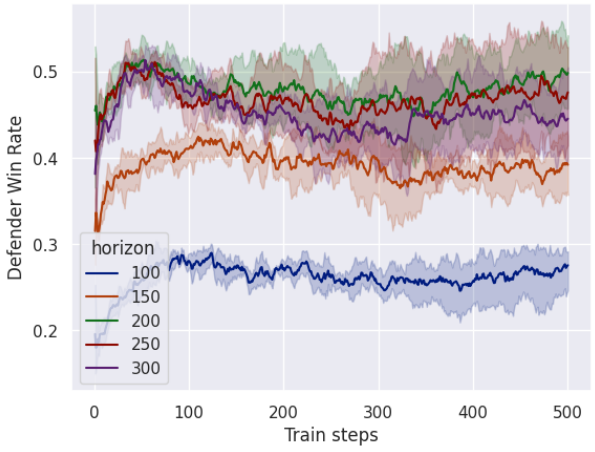

FIGURE 10. Average defender win rate for episode horizon lengths $H \in$ [100, 150, 200, 250, 300]

D. THREAT SCENARIOS

To test the robustness of the approach to various threat scenarios, we compare the defender win rate across varying values of the Impact and Exploitability scores. For the threat scenarios, we are interested in evaluating the effect of the vulnerability scores on the agent's win rate. We considered the $IScore \in [1, 2.5, 5]$ and $EScore \in [1, 2.5, 5]$. This creates nine risk scenarios as shown in Figure 11: three in the low severity range, three in the medium severity range, two in the high severity range, and the last one, when the $BaseScore = 10$ and severity is critical.

As the impact increased, as shown in Figure 11, the average defender reward increases. This occurs because the higher the impact, the more likely the defender discovers the vulnerability when the node is scanned.

As exploitability increases, the average defender win rate also increases. The higher the exploitability, the greater the *BaseScore*. Since the *BaseScore* determines the likelihood of the defender detecting a vulnerability, we expect that as the *EScore* increases the defender win rate should also increase. This is demonstrated in Figure 11. However, it's notable that the scenario with $EScore = 5$ and $IScore = 5$ does not improve significantly from scenarios with a lower *EScore*. This is likely due to the higher *EScore* which increased the capability of the attacker to compromise nodes. This is most pronounced further in training, as the curriculum introduces topologies with fewer hops between the attacker and the critical ndoe.

E. NETWORK SIZE

One of the motivations for changing the architecture of the policy and value networks is to improve scalability of the approach for larger environment network sizes. Prior work such as [6] improved performance of agents across diverse network environments but had issues with larger network sizes as the MLP used as the policy/value model did not scale.

We considered five sizes for the environment network $n \in [8, 16, 32, 64, 128]$. Figure 12 shows the win, loss, and tie rates for the sizes $n \in [8, 16, 32, 64, 128]$. The column for a network size of 128 was blacked out for the CFAD results as the approach was not tested for that value. As the network size increased, the win rate as well as the loss rate decreased. As the network size reached 32, the majority of the games ended in a tie. The results for 128 nodes are listed in Table 5 along with a comparison to similar results. Table 5 also shows a higher win rate and lower tie rate across all network sizes for the defender agent. The defender win rate increased an average of 100% while the tie rate decreased by 26%.

The attacker agent also improves its win rate by 55%, with the smallest network size showing the greatest change in the tie rate. Both the defender and the attacker agent improved overall however the greatest improvement was observed in the defender's win rate.

VI. CONCLUSIONS

We present a new intrusion response system to select defensive actions by playing a zero-sum Markov game against an attacker in a network environment. Because the proposed system was improved by utilizing a graph convolutional network (GCN) with a new reward function, it is more scalable and reliable under various attack scenarios in real network conditions. In other words, the new reward functions increase the win rates with the new agents' policy + value networks. The GCN provides scalability for the proposed approach by considering larger networks and a variety of threat scenarios.

The proposed system reflects real world network conditions because threats and threat scenarios vary from organization to organization. Our experimental results demonstrated that the proposed reward improved both reward and win

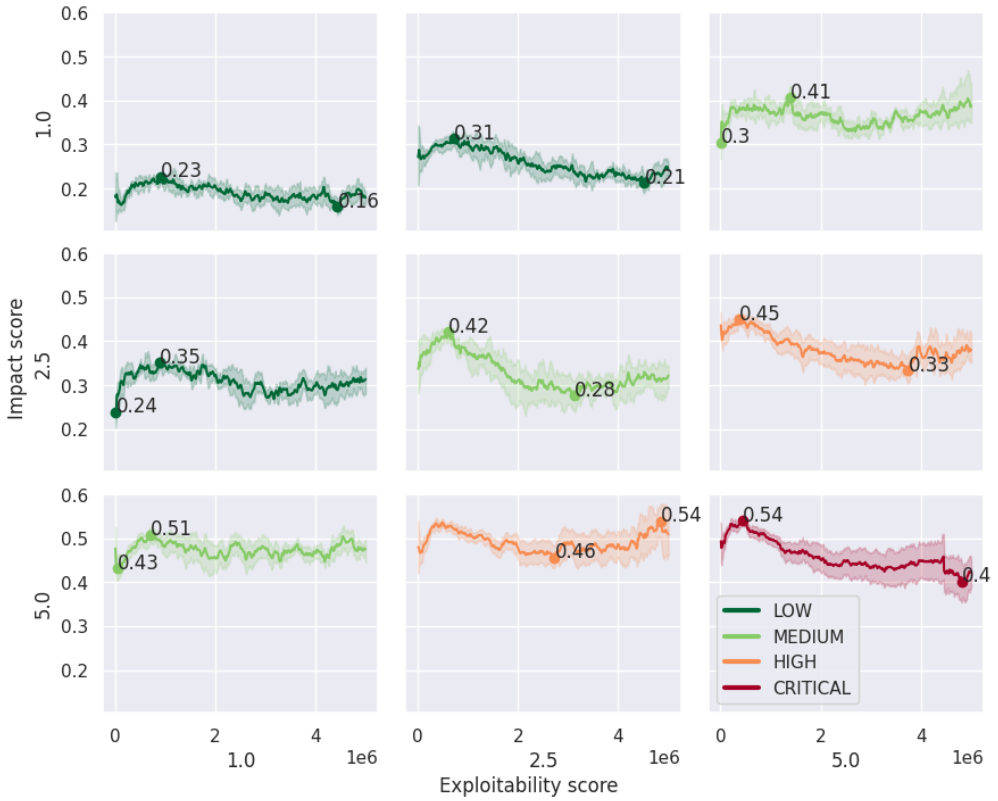

FIGURE 11. **Average defender win rate for *IScore* $\in$ [1, 2.5, 5] and *EScore* $\in$ [1, 2.5, 5] with confidence interval. The max and min values are highlighted for each cell. Colors represent the corresponding Base Score category *BS* $\in$ [*LOW*, *MEDIUM*, *HIGH*, *CRITICAL*].**

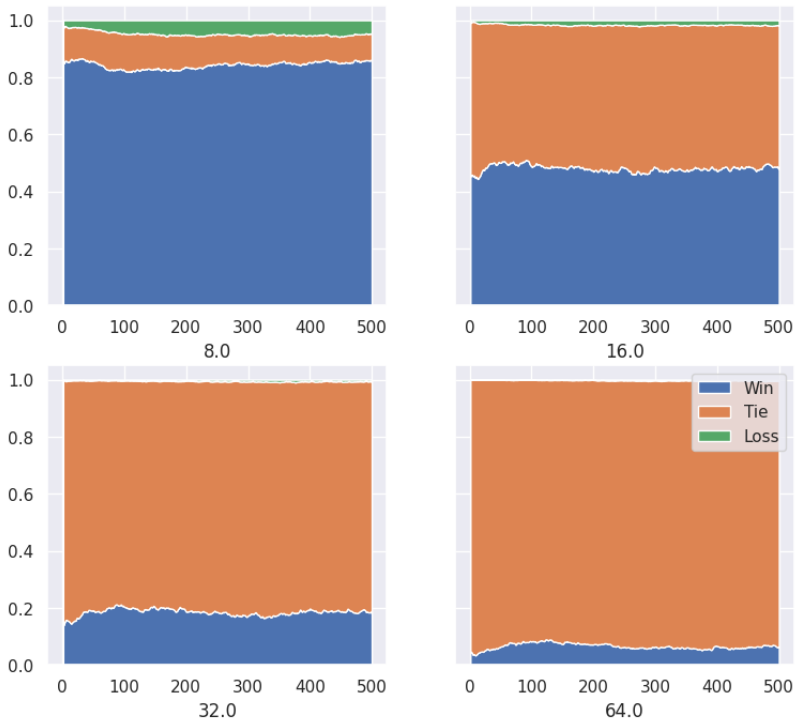

FIGURE 12. **Area plots of outcomes of games played by defender for network sizes 8, 16, 32, and 64. Win, ties, and losses are shown for all network sizes.**

TABLE 5. Network Size Results

	GNAD					CFAD [23]				
Network Size	8	16	32	64	128	8	16	32	64	128
Attacker Win Rate	25.1%	13.8%	6.5%	3.7%	1.6%	11.2%	8.5%	5.6%	3.1%	
Attacker Tie Rate	29.7%	81.9%	92.1%	95.6%	98.3%	40.3%	69.4%	87.3%	94.4%	
Defender Win Rate	84.4%	48.1%	18.6%	6.6%	2.6%	52.4%	24.5%	8.6%	2.9%	
Defender Tie Rate	10.9%	50.3%	80.9%	93.2%	97.4%	38.6%	67.8%	86.9%	94.6%	

TABLE 6. Parameter counts of GFAD models

Network Size	**8**	**16**	**32**	**64**	**128**
Parameter Count	24,780	27,348	32,484	42,756	63,300

rate for the defender. Additionally, the defender can perform well across different threat scenarios. An environment with higher *Impact* vulnerabilities is on average more favorable to the defender. The experimental results also demonstrated in an environment with vulnerabilities with higher average *Exploitability* where the defender win rate increases as the *Exploitability* increases.

VII. FUTURE WORK

We conclude with a discussion of future directions for this work. There are three main areas we identify as promising for future investigation. The first issue is the quality and quantity of training data to train agents. For automated defense methods to more generally useful, training methods should account for the lack of training data and the constantly evolving threat landscape. To bring the training environment closer to a real life scenearios, we plan to test our approach in a higher fidelity network simulation. Currently, the networking details are abstracted but they play an important role in real world threat scenarios. Furthermore, we plan to test our defensive agent against a greater variety of attacker policies. That will allow us to validate our approach across attacker strategies.

REFERENCES

[1] I. Security, "Xforce threat intelligence index 2022," https://www.ibm.com/security/data-breach/threat-intelligence, 2022.

[2] U. D. of Energy, "Colonial pipeline cyber incident," https://www.energy.gov/ceser/colonial-pipeline-cyber-incident, 2021.

[3] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "A brief survey of deep reinforcement learning," *CoRR*, vol. abs/1708.05866, 2017. [Online]. Available: http://arxiv.org/abs/1708.05866

[4] T. T. Nguyen and V. J. Reddi, "Deep reinforcement learning for cyber security," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–17, 2021.

[5] K. Hammar and R. Stadler, "Finding effective security strategies through reinforcement learning and self-play," *CoRR*, vol. abs/2009.08120, 2020. [Online]. Available: https://arxiv.org/abs/2009.08120

[6] R. G. Campbell, M. Eirinaki, and Y. Park, "A curriculum framework for autonomous network defense using multi-agent reinforcement learning," in *2023 Silicon Valley Cybersecurity Conference (SVCC)*, 2023, pp. 1–8.

[7] J. Gabirondo-López, J. Egaña, J. Miguel-Alonso, and R. Orduna Urrutia, "Towards autonomous defense of sdn networks using muzero based intelligent agents," *IEEE Access*, vol. 9, pp. 107 184–107 199, 2021.

[8] Y. Han, B. I. P. Rubinstein, T. Abraham, T. Alpcan, O. Y. de Vel, S. M. Erfani, D. Hubczenko, C. Leckie, and P. Montague, "Reinforcement learning for autonomous defence in software-defined networking," *CoRR*, vol. abs/1808.05770, 2018. [Online]. Available: http://arxiv.org/abs/1808.05770

[9] Y. Han, D. Hubczenko, P. Montague, O. De Vel, T. Abraham, B. I. P. Rubinstein, C. Leckie, T. Alpcan, and S. Erfani, "Adversarial reinforcement learning under partial observability in autonomous computer network defence," 2019. [Online]. Available: https://arxiv.org/abs/1902.09062

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:205242740

[11] R. Howard, *Dynamic programming and Markov processes*. Technology Press of Massachusetts Institute of Technology, 1960. [Online]. Available: https://books.google.com/books?id=fXJEAAAAIAAJ

[12] A. DiGiovanni and E. C. Zell, "Survey of self-play in reinforcement learning," *CoRR*, vol. abs/2107.02850, 2021. [Online]. Available: https://arxiv.org/abs/2107.02850

[13] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Machine Learning Proceedings 1994*, W. W. Cohen and H. Hirsh, Eds. San Francisco (CA): Morgan Kaufmann, 1994, pp. 157–163.

[14] M. Lapan, *Deep reinforcement learning hands-on*. Packt Publishing Limited, 2018.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

[16] 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

[17] I. Oh, S. Rho, S. Moon, S. Son, H. Lee, and J. Chung, "Creating pro-level AI for real-time fighting game with deep reinforcement learning," *CoRR*, vol. abs/1904.03821, 2019. [Online]. Available: http://arxiv.org/abs/1904.03821

[18] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. P. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *CoRR*, vol. abs/1712.01815, 2017. [Online]. Available: http://arxiv.org/abs/1712.01815

[19] B. Baker, I. Kanitscheider, T. M. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch, "Emergent tool use from multi-agent autocurricula," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. [Online]. Available: https://openreview.net/forum?id=SkxpxJBKwS

[20] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. P. Lillicrap, and D. Silver, "Mastering atari, go, chess and shogi by planning with a learned model," *CoRR*, vol. abs/1911.08265, 2019. [Online]. Available: http://arxiv.org/abs/1911.08265

[21] W. Wang, S. Jian, Y. Tan, Q. Wu, and C. Huang, "Representation learning-based network intrusion detection system by capturing explicit and implicit feature interactions," *Computers Security*, vol. 112, p. 102537, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404821003618

[22] D. K. Dake, J. D. Gadze, G. S. Klogo, and H. Nunoo-Mensah, "Multi-agent reinforcement learning framework in sdn-iot for transient load detection and prevention," *Technologies*, vol. 9, no. 3, 2021. [Online]. Available: https://www.mdpi.com/2227-7080/9/3/44

[23] R. G. Campbell, "Autonomous network defense using multi-agent reinforcement learning and self-play," Master's thesis, San Jose State University, 2022.

[24] *CVSS v3.1 Specification Document*. [Online]. Available: https://www.first.org/cvss/specification-document
[25] C.-J. Chung, P. Khatkar, T. Xing, J. Lee, and D. Huang, "Nice: Network intrusion detection and countermeasure selection in virtual network systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 10, pp. 198–211, 2013.
[26] T. Bansal, J. Pachocki, S. Sidor, I. Sutskever, and I. Mordatch, "Emergent complexity via multi-agent competition," *CoRR*, vol. abs/1710.03748, 2017. [Online]. Available: http://arxiv.org/abs/1710.03748
[27] Y. Li, D. Tarlow, M. Brockschmidt, and R. S. Zemel, "Gated graph sequence neural networks," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2016. [Online]. Available: http://arxiv.org/abs/1511.05493
[28] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, J. Gonzalez, K. Goldberg, and I. Stoica, "Ray rllib: A composable and scalable reinforcement learning library," *CoRR*, vol. abs/1712.09381, 2017. [Online]. Available: http://arxiv.org/abs/1712.09381
[29] "Ray 1.11.1." [Online]. Available: https://docs.ray.io/en/releases-1.11.1/rllib/rllib-algorithms.htmlppo
[30] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *CoRR*, vol. abs/1606.01540, 2016. [Online]. Available: http://arxiv.org/abs/1606.01540
[31] J. K. Terry, B. Black, A. Hari, L. Santos, C. Dieffendahl, N. L. Williams, Y. Lokesh, C. Horsch, and P. Ravi, "Pettingzoo: Gym for multi-agent reinforcement learning," *CoRR*, vol. abs/2009.14471, 2020. [Online]. Available: https://arxiv.org/abs/2009.14471
[32] C. Data61, "Stellargraph machine learning library," https://github.com/stellargraph/stellargraph, 2018.

ROBERT G. CAMPBELL received the B.S. and M.S. degrees in software engineering in 2022 at San Jose State University in San Jose, California.

From 2021 to 2022, he completed his masters thesis at San Jose State University under the supervision of Dr. Magdalini Eirinaki and Dr. Younghee Park. After graduation, he worked at IBM Security as a Data Scientist working on applied machine learning for cybersecurity, primarily natural language processing and computer vision. Most recently, he is providing consulting services for startups looking to apply generative AI to real world problems. His research interests include applied reinforcement learning, cybersecurity, computer vision, and natural language processsing.

MAGDALINI EIRINAKI, PH.D. Magdalini Eirinaki is a Professor at the Computer Engineering Department at San Jose State University in California. She currently also serves as the director of the MS in Artificial Intelligence program at San Jose State University. Dr. Eirinaki received her Ph.D. in Computer Science from the Department of Informatics of the Athens University Economics and Business in 2006. Her research work covers recommender systems, machine learning, data mining, social graphs, and deep learning applications. She is the recipient of the 2019 Newnan Brothers Award for Faculty Excellence, the 2017 Applied Materials Award for Excellence in Teaching and has received the SJSU distinguished faculty mentor award in 2015, 2019, 2020, 2022, and 2023.

YOUNGHEE PARK, PH.D. Younghee Park is an Associate Professor in Computer Engineering at San José State University and was a visiting professor at IBM Almaden Research in 2019. She is President and Founder of Silicon Valley Cybersecurity Institute. She received her Ph.D. in Computer Science at North Carolina State University. Her research interests lie in cybersecurity and education including machine learning, SDN/NFV security, Biometric Security, Blockchain Security, and IoT security. Her research projects have been funded by NSF, DoE, Google, and Cisco. She received the SJSU Distinguished Faculty Mentor Award in 2017 and the Faculty Award for Excellence in Scholarship in 2018. She was the Kordestani Endowed Chair at the College of Engineering Research Professor Award from 2016 to 2017. She received the Best Paper Award at ACM SIGCSE 2018 and the Best Paper Honorable Award at ACM SACMAT 2016.