Extending PNPSC Player Strategies with Continuous Firing Rates

E. Michael Bearss
Trideum Corporation
Huntsville, AL
mbearss@trideum.com

Mikel D. Petty
University of Alabama in Huntsville
Huntsville, AL
pettym@uah.edu

ABSTRACT

Organizations of various sizes and across various industries face an increasing risk from cyberattacks. Developing a deeper understanding of these attacks can facilitate the creation of more effective strategies to mitigate them. Petri Nets with Players, Strategies, and Costs (PNPSC) is an extension of Petri nets specifically designed to model cyberattacks. This formalism has been the basis for a long-running research program consisting of several interconnected research projects. Projects within that program include automatically generating PNPSC nets from the MITRE Common Attack Pattern Enumeration and Classification database of cyberattack patterns, verification and validation of the models using several complementary methods, composing multiple PNPSC nets into models of realistic computer systems, and using machine learning to improve the strategies of players present in the formalism. Previously, strategies employed by these players consisted of a limited number of rates for each transition. This limited number of rates makes it more difficult to derive those rates and reduces the fidelity of the resulting models. This paper focuses on extending the machine learning of these player strategies to use continuous transition rates. Two deep learning algorithms, Deep Q-Network and Proximal Policy Optimization make use of function approximation to effectively improve player strategies when continuous rates are present. By using continuous transition rates, SME-estimated probabilities can be used by the transitions in the models. Validated transition rates in the models are critical to ensure the models accurately represent the real-world cyberattacks described by the attack pattern. In this paper, the training performance and effectiveness of developed strategies are compared to previous research using Monte Carlo reinforcement learning.

ABOUT THE AUTHORS

E. Michael Bearss is a data scientist at Trideum Corporation, working at the Redstone Test Center in Huntsville, AL. He is a Certified Modeling and Simulation Professional and has multiple years of experience in both AI/ML and Live/Virtual/Constructive simulation. He received a Ph.D. in Computer Science from the University of Alabama in Huntsville in 2023.

Mikel D. Petty is a Professor of Computer Science at the University of Alabama in Huntsville and the Senior Scientist for Modeling and Simulation at the UAH's Information Technology and Systems Center. He has worked in modeling and simulation research and development since 1990. He received a Ph.D. in Computer Science from the University of Central Florida in 1997.

Extending PNPSC Player Strategies with Continuous Firing Rates

E. Michael Bearss
Trideum Corporation
Huntsville, AL
mbearss@trideum.com

Mikel D. Petty
University of Alabama in Huntsville
Huntsville, AL
pettym@uah.edu

INTRODUCTION

In the last few years, the number and severity of cyberattacks have grown rapidly. It is becoming common to read about major companies and government agencies affected by ransomware or data breaches. These high-profile attacks often have substantial economic impacts and may even be life-threatening. For example, in May 2021, a ransomware attack against the Colonial Pipeline caused gas shortages for a large portion of the Southeastern United States (Peñaloza, 2021). Another example is the data breach at the Office of Personnel Management in 2015. Two separate but related cybersecurity incidents caused background investigations of over 20 million federal employees and contractors to leak to the public (U.S. Office of Personnel Management, 2015). The rising frequency and severity of attacks have motivated a substantial increase in research to defend against and mitigate these cyberattacks.

The University of Alabama in Huntsville is conducting five interrelated research projects forming an integrated cyberattack modeling program. All five projects focus on an extension to Petri nets, known as Petri Nets with Players Strategies and Costs (PNPSC), used to formally model cyberattacks. Projects within that program include automatically generating PNPSC models from the MITRE Common Attack Pattern Enumeration and Classification (CAPEC) database of cyberattack patterns, verification and validation of the models using several complementary methods, composing multiple PNPSC models into models of realistic computer systems, and using machine learning to improve the strategies of players present in the formalism. PNPSC nets can model a system's dynamic states, events that occur during a cyberattack, and choices made by an attacking and defending player. PNPSC net models have been created from a wide variety of attack patterns from the CAPEC database.

The work described in this paper focuses on extending the machine learning of these player strategies to use continuous transition rates. Two deep learning algorithms, Deep Q-Network (DQN) and Proximal Policy Optimization (PPO) make use of function approximation to effectively improve player strategies when continuous rates are present. By using continuous transition rates, SME-estimated probabilities can be used by the transitions in the models. The training performance and effectiveness of developed strategies are compared to previous research using Monte Carlo reinforcement learning.

BACKGROUND

This section provides background information on several topics relevant to the rest of this paper. This section begins with an overview of Petri nets, and the extension of that formalism, of which this work is based. This section also contains a brief introduction to reinforcement learning concepts and goes into more detail on the two deep reinforcement learning algorithms used in this work.

Petri Nets

Petri nets are a graphical and mathematical modeling tool that has been widely used to model a variety of systems. Petri nets are a type of directed, weighted, bipartite graph consisting of two types of nodes, places, and transitions. Arcs are used to connect transitions to places and places to transitions. Arcs cannot exist between two places or two transitions. Each arc is given a positive integer weight, where a *k*-weighted arc can be interpreted as a set of *k* parallel arcs. A place with an outgoing arc connected to a transition is an input place of that transition, whereas a place with an incoming arc connected to a transition is an output place of that transition (Murata, 1989).

Each Petri net place can contain a discrete number of tokens, ranging from zero to a maximum value defined by a parameter. The number of tokens present in each place is known as the Petri net's marking. A transition of a Petri net is called *enabled* if there are greater than *k* tokens in all its input places, *k* being the weight of the arc. Figure 1 shows an example Petri net consisting of three places and a single transition. Places p_1 and p_2 are input places of transition t_1 and are connected with arcs to t_1 with weights two and one, respectively. Place p_3 is an output place of t_1, also connected with an arc weight of one. Because all input places of t_1 have a number of tokens greater than or equal to their arc weight, transition t_1 is *enabled*.

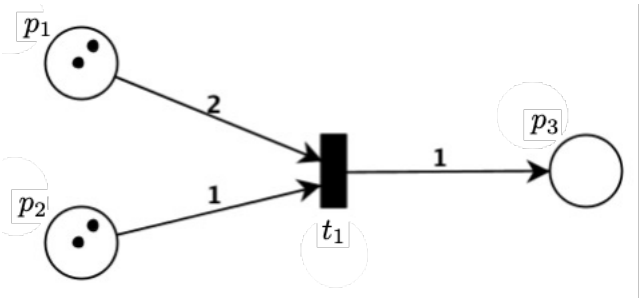

Figure 1. An example Petri net

During execution, one of the enabled Petri net transitions is selected at random to fire. When the selected transition *fires*, tokens are removed from each input place and added to each output place. The number of tokens added or removed is equal to the weight of the arc connecting the transition. Thus, the total number of tokens in the net does not necessarily remain constant.

The Petri net formalism is defined by the following 6-tuple, $PN = (P, T, W, B, M_0, L)\ where$:

1. $P = \{p_1, p_2, \dots\}$; finite, non-empty set of places
2. $T = \{t_1, t_2, \dots\}$; finite, non-empty set of transitions
3. $W \subseteq (P \times T) \cup (T \times P)$; set of arcs from places to transitions and transitions to places
4. $B: P \to Z^+ \cup \infty$; upper bound on tokens per place
5. $M_0: P \to Z^+$; initial marking of tokens in places, with $0 \leq M_0(p) \leq B(p)$ for every $p \in P$
6. $L: W \to Z^+ \cup \{-1\}$; $L: W \ \ Z^+ \cup \{-1\}$; arc weights

Petri Nets with Players Strategies and Costs

The PNPSC formalism is an extension of the Petri net formalism and was explicitly developed to model cyberattacks. The formalism extends Petri nets by adding players and transition rates controllable by those players. Whereas in a Petri net, enabled transitions are chosen at random to fire, in a PNPSC net, the transition rates defined by the formalism are considered. Each transition t_i has a firing rate of λ_i. A random variate is generated from an exponential distribution using λ_i as the exponential distribution's rate parameter λ. The enabled transition with the smallest random variate, which corresponds to the smallest interfering time, is fired. The firing rates are then updated for the new marking of the Petri net, and the cycle is repeated.

Another important modification to the original Petri net formalism is the addition of inhibitor arcs. Inhibitor arcs may be present from a place to a transition. If the input place of an inhibitor arc is marked, it prevents the connected transition from firing, even if the transition is otherwise enabled (Agerwala & Flynn, 1973). In the case of modeling cyberattacks, inhibitor arcs provide the capability for a player to disable the action of another player.

Players attempt to influence the sequence of transition firings during the execution of a PNPSC net by updating the rates under their control. Each time a player updates one of these rates, they may incur a cost related to the change of the rate. The transitions controlled by the player represent the actions available to that player that may influence the outcome of the cyberattack. The mapping of player observable places and changes to the player's controlled rates is known as the player's strategy.

PNPSC nets are formally defined by the following 14-tuple $PNPSC = (P, T, W, M_0, B, L, G, \Theta, O, F, \Omega, \Gamma, C, D)$:

1. P, T, W, M_0, B, L as defined for a standard Petri net
2. $G = \{g_1, g_2, \ldots\}$ finite, non-empty set of players
3. $\Theta = (T_o, T_1, T_2, \ldots, T_{|G|})$; partition of transition set T into $|G| + 1$ subsets such that $\Theta = T_0 \cup T_1 \cup T_2 \cup \ldots \cup T_{|G|}$ and $T_j \cap T_k = \varnothing$ for $0 \leq$ j, k $\leq |G|$ and j $\neq$ k;
   $T_i =$ set of transitions controlled by player g_i for $1 \leq i \leq |G|$ and
   $T_0 =$ set of stochastic transitions not controlled by any player
4. $O = (O_1, O_2, \ldots, O_{|G|})$; collection of $|G|$ subsets of place set P, i.e., $O_i \subseteq$ P for $1 \leq$ i $\leq |O|$
   O_i is the subset of place set P observable by player g_i
5. $F: T_0 \rightarrow \mathbb{R}^+$;fixed firing rates for non-player-controlled transitions
6. Ω: (T $- T_0$) $\rightarrow (\mathbb{R}^+ \times \mathbb{R}^+)$; initial and maximum firing rates for player-controlled transitions
7. $\Gamma : (\Gamma_1, \Gamma_2, \ldots, \Gamma_{|G|})$; collection of functions $\Gamma_i: {M^*}_{O_i} \rightarrow \mathbb{R}^{+|T^i|}$ where Γ_i is a mapping from the possible markings of player g_i's observable places to the desired firing rates for each of player g_i's controlled transitions
8. $C = (C_{fire}, C_{change})$; where C_{fire}: $(T \rightarrow \mathbb{R}^+)$ is the cost for firing a transition and C_{change}: $(T \times \mathbb{R}^+) \rightarrow \mathbb{R}^+$ is the cost for changing the rate of a transition by $\delta \in \mathbb{R}^+$
9. $D: T \rightarrow \wp$(G); players that incur a cost for a fired or changed transition

A tool has been created to automatically generate PNPSC nets to model cyberattacks described in the CAPEC database (Whitaker, 2019). The PNPSC net models the steps in the execution flow of the cyberattack pattern and the various techniques an adversary can use to accomplish those steps. Each model can be divided into four phases: *Explore*, *Experiment*, *Exploit*, and *Goals*. Each of these phases is connected by a "pivot" transition representing the adversary progressing through that phase of the cyberattack. The PNPSC net for CAPEC-63 Cross-Site Scripting is shown in Figure 3. Descriptions of each transition and their corresponding initial rates are shown in Table 3 – Table 6 located in the Appendix. A long-running research program based on PNPSC net models of cyberattacks, of which the work described in this paper is a part, is summarized in (Petty, 2022).

Reinforcement Learning

Reinforcement learning agents learn to perform a task by interacting with their environment. This interaction is usually modeled as a Markov Decision Process (MDP) and is shown in Figure 2 (Sutton & Barto, 2018). By repeatedly interacting with the environment and receiving rewards, the agents can learn the best actions to take to maximize the long-term reward.

The long-term reward is also called the expected value of a state. A discount factor, γ, can also be applied to reduce the value of future rewards. This mapping of states to expected returns is known as a value function. By using dynamic programming, an agent is able to iteratively update the expected return by sampling the environment. An optimal policy can be found using the Bellman optimality equation.

An agent uses a policy to select the next action (typically choosing the action that leads to the highest expected reward). One common policy used by learning agents is the ε-greedy policy, in which agents choose the highest valued action with probability $1 -$ ε and a randomly selected action with probability ε.

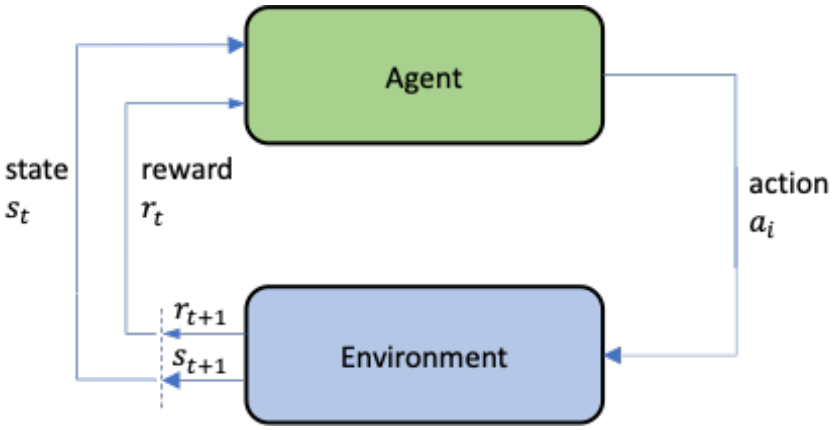

Figure 2. Agent-environment interaction in an MDP

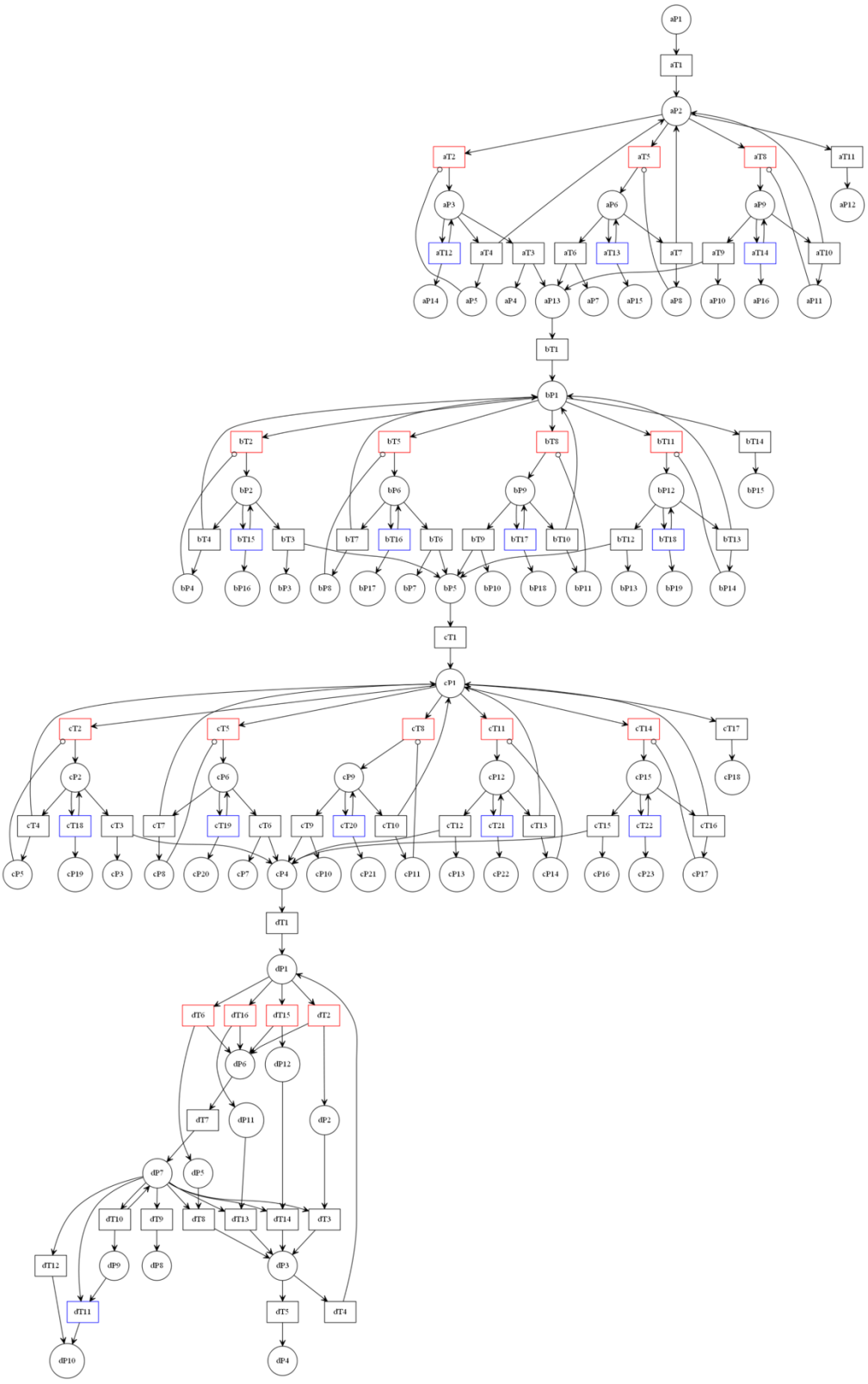

Figure 3. PNPSC net to model CAPEC-63 Cross-Site Scripting

Deep Q-Learning

Deep Q-learning is a type of reinforcement learning algorithm that uses deep neural networks to learn a policy that maximizes the long-term expected reward. It is a combination of Q-learning and artificial neural networks, which allows the algorithm to use function approximation to estimate the values of unexplored states and actions. Deep Q-learning has been demonstrated to reach superhuman skill levels in a variety of games (Mnih et al., 2013) (Marvin et al., 2019). One version of the deep Q-learning algorithm is shown in Figure 4. This particular version of the algorithm makes use of experience replay, a technique that allows the agent to store experience and learn from it multiple times.

Figure 4: Deep Q-learning with Expereience Replay

Input: initial policy parameters θ_0, and value function parameters ϕ_0

for *episode=1,2,...,M* **do**
  Initialize sequence $s_k = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$.
  for *t=1,2,...,T* **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = max_a\, Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma\, max_{a'}\, Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$
  end for
end for

Proximal Policy Optimization

Proximal Policy Optimization (PPO) is a family of policy gradient methods of reinforcement learning. PPO algorithms have the sample efficiency and reliability of Trust Region Policy Optimization (TRPO) while being much simpler due to their use of only first-order optimization. PPO has been shown empirically to perform as well as TRPO (Schulman et al., 2017). There are two primary variants of the PPO algorithm: PPO-Penalty and PPO-Clip. PPO-Penalty performs a KL-constrained update like TRPO but penalizes the KL-divergence in the objective function rather than using it as a constraint. PPO-Clip forgoes the KL-divergence term, instead relying on specialized clipping in the objective function. In this work, the PPO-Clip algorithm is used, the pseudocode for which is shown in Figure 5 (Achiam, 2020).

PPO is an on-policy method that works by alternating between sampling data from the environment and optimizing a surrogate objective function. The amount of randomness in the policy is controlled by the initial conditions and the training procedure. As training occurs, the update rule encourages the algorithm to exploit its current knowledge, and the policy becomes less stochastic.

Figure 5: PPO-Clip

Input: initial policy parameters θ_0, and value function parameters ϕ_0

for $k=0,1,2,...$ **do**

Collect set of trajectories $\mathcal{D}_k=\{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.

Compute rewards-to-go $\hat{R}_t$.

Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.

Update the policy by maximizing the PPO-Clip objective: $\theta_{k+1} =$

$$*arg\,max_\theta \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} min(\frac{\pi_0(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t,a_t), g(\epsilon, A^{\pi_{\theta_k}}(s_t,a_t))),$$

typically via stochastic gradient ascent with Adam.

Fit value function by regression on mean-squared error:

$$\phi_{k+1} = *arg\,min_\theta \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} (V_\phi(s_t) - \hat{R}_t)^2,$$

typically via some gradient descent algorithm.

end for

EXPLANATION OF LEARNING AGENTS

The environment used in this work to train the learning agents uses a combination of Monte Carlo tree search (MCTS) and deep reinforcement learning methods. MCTS is a type of reinforcement learning method used for finding optional decisions in a game tree by randomly sampling the given environment. Many successful AI game-playing algorithms, such as AlphaGo, have incorporated MCTS to beat the best human players in games with extremely high branching factors. By using a combination of these two reinforcement learning methods, much of the stochasticity of PNPSC execution can be reduced. Additional information about the environment used to train the agents can be found in (Bearss & Petty, 2023).

Two separate implementations were developed to demonstrate the ability of the learning agents to improve player strategies for PNPSC nets containing continuous transition rates. A DQN agent was implemented that performs discrete actions to update continuous rate values. A second implementation using the PPO algorithm was also developed. This algorithm is capable of simultaneously updating the rates of all controlled transitions to any desired values.

To quickly integrate different reinforcement learning algorithms to improve PNPSC net player strategies, a PNPSC simulator compliant with the Gym application programming interface (API) was developed. Gym provides a standard API to communicate between learning algorithms and environments (OpenAI, 2021). Once a Gym environment is developed, many off-the-shelf reinforcement learning algorithms can quickly be applied. In addition, Gym environments are able to be easily augmented through the use of *wrappers.* Gym wrappers can change the state or action space, or change the behavior of the environment. Monitoring tools commonly work by wrapping the environment, allowing the ability to easily track the training performance of the learning agents.

To conform to the Gym standard API, the environment must provide an observation and action space. In this instance, the agent models a particular player strategy (Γ in the PNPSC formalism). The observation space consists of the number of tokens present in each place observable to the player and the rate of each transition controlled by the player. The action space consists of the desired rates for each transition under the player's control. The agent must only interact with the environment using the ***reset*** and ***step*** functions. The ***reset*** function returns the PNPSC simulation to the initial state (initial marking and rates defined in the formalism) and provides the agent with its observation consisting of the current observable marking and controllable rates. The ***step*** function allows the agent to update the transition rates it controls and receive its next observation and reward. The Gym environment is designed to closely correspond with the MDP shown in Figure 2. Because there may be more than one player

present in the formalism, other players may make rate updates between the agent's call to ***step*** and receiving the next observation. These other player interactions will be treated as dynamics of the environment. By convention, the agent is not allowed to obtain any information outside of the observation returned by the ***reset*** and ***step*** functions.

DQN, which is a discrete control algorithm, requires an enumerable action space. A wrapper can be added to adjust the transition rates based on a selection of possible actions (e.g., replace the rate of aT2 with the value 10). Each action available to the DQN agent applies a specified function and value to one of the existing transition rates. Although it is possible to use a different function for each action, all DQN agents in this work use a single function for all actions for simplicity. This versatile implementation allows for many different strategies to be easily developed. For example, to approximate the performance of the Monte Carlo agents, the function could replace the current rate with the specified value. This would allow the DQN agent to update the specified transition rate to either 0 or 10 directly. For the DQN agents used in this work, the current rate was multiplied by the specified value. The action values consisted of 0, 0.5, 2, and 4. The DQN agent could half, double, or quadruple a transition rate or set it to 0 to disable the transition entirely. If the agent attempts to increase a rate above the maximum value, it is clipped to that value. Restricting the number of values for each transition, and thus the number of actions available to the DQN agent is important to ensure the effective learning of the agent, as demonstrated by the work described in (Bearss et al., 2022). Allowing the attacker four possible values for each attacker-controlled transition (plus an additional *no-update* action) gives the DQN agent a total of $16 \cdot 4 + 1 = 65$ actions for CAPEC-63.

One notable limitation of discrete control algorithms is their ability to only perform a single action each time they interact with the environment (Mnih et al., 2013). This limitation can prevent the learning agents from developing an optimal strategy for most PNPSC nets. Due to the game-like nature of the execution of PNPSC nets, this could be seen as a special case of the formalism where all players are limited to a single transition rate update per transition fire. This would force players to take turns updating rates rather than allowing them to update the entire set of transition rates whenever desired.

There are several methods to allow DQN agents to emulate taking multiple actions in each interaction with the environment. However, they all have drawbacks. Rather than selecting a single action, the DQN agent can select from a set of action combinations. If the DQN agent for the attacker player for CAPEC-63 is allowed to choose any combination of two actions, this leads to $\binom{16}{2} \cdot (5 \cdot 5 - 1) + 1 = 2881$ possible combinations. For each combination of two transitions, there are five possible rates for each transition (1 has been added as a rate option to allow the player only to update one transition). The -1 term in the multiplier is present to remove the (1, 1) case. This case has the same result for every combination of transitions and is equivalent to the *no-update* action. Allowing the DQN agent to combine more than two actions per transition firing is not practical for any of the tested PNPSC nets representing CAPEC attack patterns because of the very large number of resulting combinations.

Another method to emulate multiple simultaneous actions is to allow the DQN agent to interact with the environment multiple times before the next transition is selected to fire. This method removes the dramatic expansion of the action space but requires a much more sophisticated policy to obtain good performance. When there are no other players in the PNPSC net, an agent can reliably choose a transition rate to update based solely on the current observable marking. However, when the agent chooses a second transition rate to update, it must now also consider the rates of the transitions under its control. Due to the Markov property, the agent can not "remember" which transition rates it has updated. The agent must observe this from the state space. As the number of updates per transition firing increases, the policy network must become increasingly complex.

Both methods to allow the DQN agent to perform multiple actions per environment interaction were tested. When tested with a small PNPSC net, such as only the Explore phase of CAPEC-63, both implementations showed improvement over the single-action agent. However, when tested with the full PNPSC net for CAPEC-63, only the multiple-action agent was able to perform better than the single-action agent. In addition, this agent required a significantly more complex policy to perform well. The PPO agent is capable of multiple transition rate updates per transition fire with greater stability and less complexity. Due to these limitations, all DQN agents evaluated in this work were restricted to a single transition rate update per transition fire.

A PPO agent was also implemented that could directly manipulate the transition rates under the player's control. This agent was able to update all transition rates simultaneously. The output of the agent's policy network was added to the existing rates (e.g., an output of [1, -3] increases the rate of transition 1 by 1 and decreases the rate of

transition 2 by 3). If the agent tries to increase the rate above the maximum or below the minimum, the rate is clipped to that value. The agent does not incur an additional cost if it attempts to adjust the transition rate outside its allowed range.

To allow for an objective comparison with the Monte Carlo agents described in (Bland, 2018), the minimum rate of 0 and maximum rate of 10 were maintained. The cost function for the attacker player for all attack patterns was also identical, shown in (1). The structure of the PNPSC nets and the transition rates used were also identical.

$$cost = \frac{\sum \Delta rate}{10} \quad (1)$$

RESULTS

The DQN and PPO agents, explained in the previous section, were trained to improve the attacker's and defender's strategies for three different CAPEC attack patterns: CAPEC-63 Cross-Site Scripting, CAPEC-66 SQL Injection, and CAPEC-163 Spear Phishing. In this paper, only information on the attacker player for CAPEC-63 is presented, more information about the other players and attack patterns can be found in (Bearss, 2023). This section contains two subsections, the first of which utilizes a linear cost function to compare to previous work using machine learning to optimize the player strategies. The second subsection replaces this cost function with a more complex piecewise function that contains an exponential term to encourage the learning agents to use less extreme rate updates (not always choosing the minimum or maximum rate).

Linear Cost

Figure 6 shows a comparison of the strategy performance of the DQN and PPO agents for the CAPEC-63 PNPSC net. The graphs in this section show the mean of all previous episode rewards. This is done to help illustrate the highly stochastic nature of PNPSC net execution. Each agent was evaluated 100 000 times.

Each graph in this section also shows a static agent that performs no action and an agent developed with Monte Carlo methods with only two possible rates, 0 and 10. The Monte Carlo agents were developed for each CAPEC attack pattern and player combination from the work described in (Bland, 2018). The strategy of the Monte Carlo agent gives an expected upper bound on the performance of the newly developed learning agents. Similarly, the strategy of the static agent provides a lower bound on expected performance. A learning agent performing reliably better than the static agent demonstrates some level of *learning*.

For this model, both the DQN and PPO agents were trained using a parallel environment count of 64 (the number of simulations to determine the score of a game state). For more information about the environment, see (Bearss & Petty, 2023). The DQN agent was trained for 200 000 steps (one interaction with the environment) taking approximately 121 minutes. The PPO agent was trained for 1 500 000 steps taking approximately 478 minutes. For comparison, the Monte Carlo agents were trained for 4 500 000 steps, with most of the learning occurring in the first 900 000 steps (Bland, 2018). Note that the samples measured in this work refer to the training environment using the game tree algorithm described in (Bearss & Petty, 2023). Each interaction with the training environment results in many steps being executed in the original PNPSC simulator algorithm. When trained with the original PNPSC simulator, the deep reinforcement learning agents are significantly less sample efficient.

Due to the neural network used to store the learning agent's policy, as well as the large state and action space, it is difficult to enumerate the entire strategy developed by the agent. In addition to the probabilistic policy, each possible marking can have a different rate adjustment. To illustrate the strategy of the players, a single PNPSC execution is shown for the DQN agent in Table 1, with the marking and associated rate updates made during the execution. For the current observable marking listed in the *Marking* column, each rate of the transition listed in the row is increased by the value to its right (e.g., in Table 1, if only place aP2 is marked, the rate of transition aT8 is increased by 8). To maintain consistency between the DQN and PPO agent strategies, the rate increase (with negative values representing reductions) for each transition is shown.

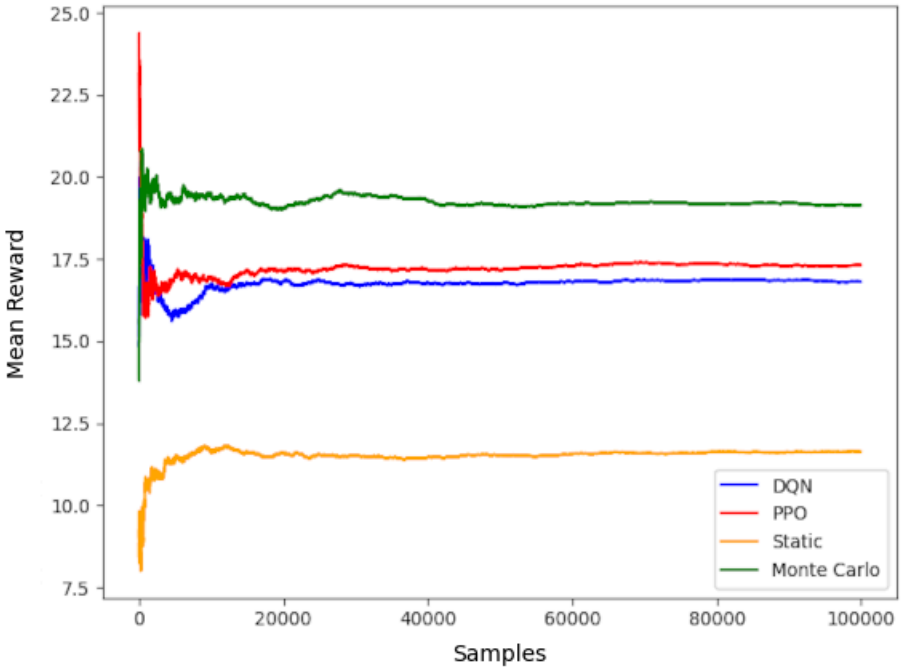

Figure 6. CAPEC-63 attacker strategy performance with a linear cost function

Table 1. CAPEC-63 sample DQN attacker strategy

Marking	Strategy
aP2	aT8 +8
aP10	bT5 +1
aP10, bP1	cT2 +4
aP10, bP10	cT8 +5

Exponential Cost

One particular aspect of the existing research of PNPSC player strategy improvement makes the developed strategies in the previous subsection somewhat uninteresting. If the PNPSC net representing an attack pattern phase is viewed as a tree (ignoring cycles), each group of places and transitions representing a technique (or failing) can be viewed as a branch. In all three PNPSC nets, some branches lead to a smaller reward received by the player, whereas others lead to a larger one. For example, in Figure 7, aP12, the output place of transition aT11 is not the input to any other transitions. This transition firing represents the attacker giving up and ends the net's execution. Because increasing the rate of aT2, aT5, and aT8 all decrease the chance of aT11 firing, they all improve the strategy of the attacker.

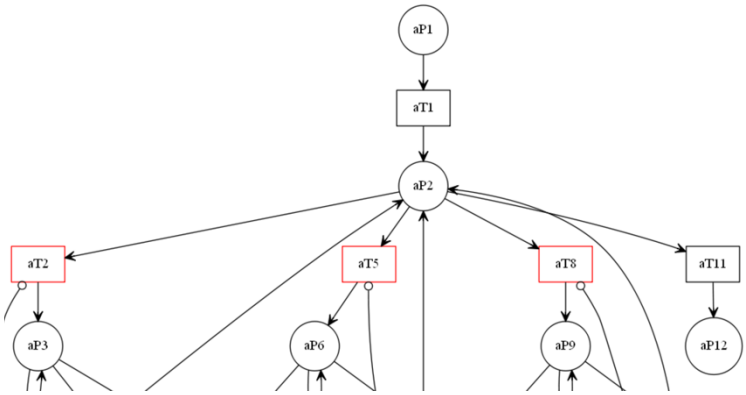

Figure 7. Optimal strategies will increase the rates of aT2, aT5, and aT8

More interesting strategies can be encouraged by using a different cost function. A new cost function, shown in (2), has two notable properties. First, this piecewise function has an exponential cost to increase the rate and a linear one to decrease it. In many situations, there is a concept of diminishing returns, where investments made to improve performance do not scale linearly. The second property is the conservation of cost. This property maintains the same cost of updating a transition regardless if the update is performed in one step or many steps that sum to that amount (if all updates are in the same direction). This property is important to ensure that the PPO algorithm is unable to exploit the exponential behavior of the cost function by performing many small updates.

$$f(r', r) = \begin{cases} r - r' & r' \leq r \\ 1.75^{r'} - 1.75^{r} & r' > r \end{cases} \qquad (2)$$

$$cost = \frac{\sum f(\text{updated rate, current rate})}{10}$$

The same two learning agents were retrained using the same environment with only the cost function modified. As with the previous subsection, the graphs in this subsection also show a static agent and a Monte Carlo agent derived from the work described in (Bland, 2018). However, because the Monte Carlo agents were trained using different cost functions, and thus have poor estimates of the cost of updating transition rates, they perform poorly in this environment.

The performance of the DQN and PPO agents for the attacker player of CAPEC-63 is shown in Figure 8. In this case, even after 500 000 steps, the DQN agent is unable to find a strategy that reliably outperforms the static player. However, the PPO agent's use of continuous transition rates allows it to achieve significantly better performance. The PPO agent was trained for 1 000 000 steps. The strategy of the PPO agent is shown in Table 2.

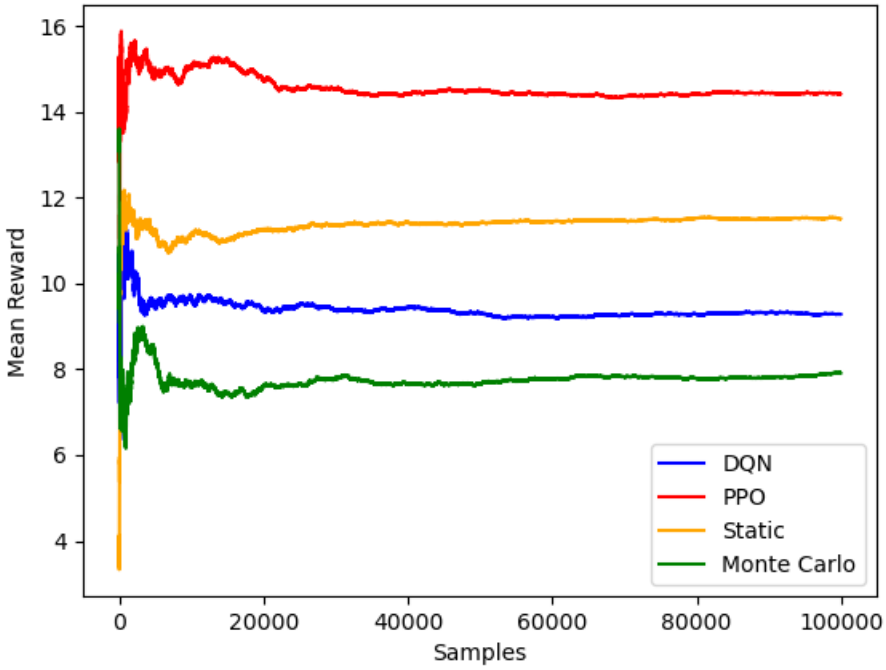

Figure 8. CAPEC-63 attacker strategy performance with an exponential cost function

Table 2. CAPEC-63 exponential cost sample PPO attacker strategy

Marking	Strategy	
aP1	aT2 +4 aT8 +5.5 cT2 +2.25 cT14 +0.75	aT5 +1 bT5 +1 cT5 +1.5
aP2	cT5 +1.5	
aP2, aP11	bT2 +1.75 cT5 +0.25 dT6 +2.5	bT8 +1.5 cT14 +0.5
aP11	aT8 +0.5 bT8 +2.25 cT2 +1.75 dT6 +0.25	bT2 +0.25 bT11 +0.25 cT14 +0.75 dT15 +0.5
aP4, aP11	bT2 +0.25 dT6 +0.75	bT8 +0.25
aP4, aP11, bP1	bT11 +0.5 dT2 +0.5 dT16 +1	cT14 +0.25 dT15 +0.75
aP4, aP11, bP10	bT5 +0.25 dT6 +0.25	cT14 +0.75
aP4, aP11, bP10, cP1	cT18 +0.25 dT5 +0.5	cT14 +0.25
aP4, aP11, bP10, cP3	dT15 +1	
aP4, aP11, bP10, cP3, dP1	dT15 +0.25	

CONCLUSION

The work described in this paper was able to effectively improve the player strategies for several PNPSC nets when continuous transition rates were used. A DQN agent able to perform a single transition rate update to several possible values, as well as a PPO agent able to update all player-controlled transition rates simultaneously, were developed and demonstrated to improve player strategies on a variety of PNPSC nets representing attack patterns from the CAPEC database of cyberattack patterns.

The strategy performance of these learning agents was compared to existing work that used Monte Carlo reinforcement learning to improve strategies when only two possible rates for each transition were used. Because every combination of rates was tried at least once, it is expected that the Monte Carlo method will produce the optimal strategy when constrained to only these choices. As explained earlier this is also expected to be the optimal performance when the cost function in (1) is used. As expected, the performance of both the DQN and PPO agents fell between the static strategy and the strategy used by the Monte Carlo learning agent. When a cost function with an exponential term was introduced, the advantage of continuous transition rates became clear. The DQN agent was unable to successfully improve the player's strategy, however, the PPO agent showed significant gains.

REFERENCES

Achiam, J. (2020). Proximal policy optimization. Retrieved from https://spinningup.openai.com/en/latest/algorithms/ppo.html

Agerwala, T. & Flynn, M. (1973). Comments on capabilities, limitations and "correctness" of Petri nets. ACM SIGARCH Computer Architecture News, 2(4), 81-86.

Bearss, E. M. (2023). Extending machine learning of cyberattack strategies with continuous transition rates (Doctoral dissertation). University of Alabama in Huntsville.

Bearss, E. M., Alonso Guzman, J. G., Dang, V., Regmi, S., Roark, N., & Petty, M. D. (2022). Using machine learning to win Strike Force One. Paper presented at AlaSim 2022.

Bearss, E. M. & Petty, M. D. (2023). A Deep Reinforcement Learning Technique for PNPSC Net Player Strategies. In Proceedings of the 2023 ACM Southeast Conference (pp. 96-103). ACMSE 2023. Association for Computing Machinery.

Bland, J. A. (2018). Machine learning of cyberattack and defense strategies (Doctoral dissertation).

Mavrin, B., Zhang, S., Yao, H., Kong, L., Wu, K., & Yu, Y. (2019). Distributional Reinforcement Learning for Efficient Exploration. *arXiv [Cs.LG]*. Retrieved from http://arxiv.org/abs/1905.06125

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing Atari with deep reinforcement learning. Retrieved from https://arxiv.org/abs/1312.5602

Murata, T. (1989). Petri Nets: Properties, Analysis and Applications. Proceedings of the IEEE, 77, 541-580. https://doi.org/10.1109/5.24143

OpenAI. (2021). Openai/gym: A toolkit for developing and comparing reinforcement learning algorithms. Retrieved from https://github.com/openai/gym

Peñaloza, M. (2021, May). Ransomware attack shuts down a top U.S. gasoline pipeline.

Petty, M. D., Whitaker, T. S., Bearss, E. M., Bland, J. A., Cantrell, W. A., Colvett, C. D., & Maxwell, K. P. (2022). Modeling Cyberattacks with Extended Petri Nets: Proceedings of the 2022 ACM Southeast Conference. Retrieved from https://dl.acm.org/doi/abs/10.1145/3476883.3520209

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. Retrieved from https://arxiv.org/abs/1707.06347

Sutton, R. S. & Barto, A. G. (2018). Reinforcement Learning: An Introduction. MIT Press.

U.S. Office of Personnel Management. (2015). Cybersecurity resource center cybersecurity incidents. Retrieved from www.opm.gov/cybersecurity/cybersecurity-incidents

Whitaker, T. S. (2019). Generating cyberattack model components from an attack pattern database (Doctoral dissertation).