Deep Reinforcement Learning for Cyber System Defense under Dynamic Adversarial Uncertainties

Ashutosh Dutta[1], **Samrat Chatterjee**[1], **Arnab Bhattacharya**[1], **Mahantesh Halappanavar**[1]

[1]Pacific Northwest National Laboratory
902 Battelle Boulevard
Richland, Washington 99354, USA

Abstract

Development of autonomous cyber system defense strategies and action recommendations in the real-world is challenging and includes characterizing system state uncertainties and attack-defense dynamics. We propose a data-driven deep reinforcement learning (DRL) framework to learn proactive, context-aware, defense countermeasures that dynamically adapt to evolving adversarial behaviors while minimizing loss of cyber system operations. A dynamic defense optimization problem is formulated with multiple protective postures against different types of adversaries with varying levels of skill and persistence. A custom simulation environment was developed and experiments were devised to systematically evaluate the performance of four model-free DRL algorithms against realistic, multi-stage attack sequences. Our results suggest the efficacy of DRL algorithms for proactive cyber defense under multi-stage attack profiles and system uncertainties.

Introduction

Decision support for cyber system defense in real-world dynamic environments is a challenging research problem that includes the dynamical characterization of uncertainties in the system state and the incorporation of dynamics between attackers and defenders. Cyber defenders typically operate under resource constraints (e.g., labor, time, cost) and must update their strategies and tactics dynamically as the system evolves with/without attack influence from adaptive adversaries. Often, the defender may be unaware of system compromise and must adopt proactive strategies to maintain mission-critical operations. The past decade has seen a growing body of literature focused on the application of game-theoretic approaches for cybersecurity (Roy et al. 2010; Liang and Xiao 2012). These approaches typically involve resource allocation optimization with Markov decision process (MDP) models in non-cooperative settings. Cyber system attack and defense modeling methods also include: 1) probabilistic approaches for system reliability and attack outcome dependency (Hu et al. 2017; Chen, Xu, and Shi 2018); 2) Bayesian networks (Frigault et al. 2008; Shin et al. 2015); and 3) fault/decision trees (Fovino, Masera, and De Cian 2009). Although attack graph-based methods (Poolsappasit, Dewri, and Ray 2011) for identifying system vulnerabilities and known exploits are valuable, within realistic settings, insufficient and imperfect information about system properties and attack goals are typically available to the defender.

Recent advances in reinforcement learning (RL) approaches have led to the development of partially observable stochastic games (POSG) in partial information settings (MacDermed, Isbell, and Weiss 2011; Sutton and Barto 2018). Also, cyber system state-space modeling has generated interest in potential use of POSGs for cybersecurity problems (Ramuhalli et al. 2013; Chatterjee et al. 2016; Tipireddy et al. 2017). However, POSGs tend to be general formulations and are often intractable problems. In addition, the state-of-the-art in cyber decision-support also includes the use of partially observable Markov decision process (POMDP) models, as well as distributed POMDPs, for solving a variety of problems such as: 1) cyber risk assessment (Carin, Cybenko, and Hughes 2008), 2) uncertainty in penetration testing (Sarraute, Buffet, and Hoffmann 2012), 3) network task time allocation (Caulfield and Fielder 2015), 4) data exfiltration detection (McCarthy et al. 2016), and 5) effective deception resource allocation (Islam et al. 2021). Most of these problems either focus only on specific attacks (or attack types), consider a limited defender action space, and assume that a system model is available. Recent work in deep RL (DRL) methods for cybersecurity primarily focus either on optimizing network operations or cyber defense against different threat types (Nguyen and Reddi 2021; Dutta, Al-Shaer, and Chatterjee 2021). While progress mostly includes focus on model-free DRL methods, there is a critical need for investigating further the role of different DRL algorithms for cyber defense trained under diverse adversarial settings.

This paper focuses on the applicability of DRL in optimizing cybersecurity defense planning against strategic multi-stage adversaries. Specifically, the objective of a DRL defense agent is to compute context-aware defense plans by learning network and multi-stage attack behaviors while minimizing impacts to benign system operations. Generally, the dynamics of cyber systems depend on many correlated factors (e.g., traffic volume, network utilization) which may exhibit uncertain behaviors across time due to unanticipated

physical link failures, sensor errors, and others. On the other hand, a stealthy cyber adversary may adapt their strategies (i.e., tactics and techniques) based on current network conditions and ongoing attack impact. For example, multi-stage attacks can propagate through the deployment of multiple software processes (e.g., Application Programming Interface (API) calls) to execute attack actions without inducing suspicion, instead of relying on a single process. However, a defender may not always be able to block a process due to its role in maintaining critical operations governed by interdependent processes. Thus, defense planning needs to consider the aggregated impact of different attack actions at different stages of its propagation. Figure 1 presents a multi-stage attack propagation template based on MITRE ATT&CK framework (MITRE 2022) that was leveraged in our study for evaluating multiple DRL-based defense mechanisms. Within an attack-defense interaction, an adversary can start from any technique of the *Reconnaissance/Initial Access* tactic, and wins if they reach any technique of *Impact/Exfiltration* tactic. Based on defense actions, the attacker may abort (i.e., move to the *Attack Terminated* state or defender's win) or persist to move on to the next stage.

The main contribution of this paper is to systematically evaluate the performance of multiple DRL algorithms for cyber defense that were trained under diverse adversarial uncertainties. The next section briefly describes DRL algorithms used in this study. Next, we describe the proposed autonomous cyber-defense framework and custom simulation environment. This is followed by experimental results and discussion, and some concluding remarks.

Background

Cyber defense DRL agents compute a policy that recommends optimal action at the current cyber-network state. At time-sequence t, the agent executes an action a_t at current state s_t and receives reward $R(s_t, a_t)$ that is used to update the policy. One of the objective of DRL agent is to balance the trade-off between exploration (i.e., executing random actions to understand consequences) and exploitation (i.e., executing optimal actions based on previous exploration knowledge). This research uses ϵ-greedy approaches, where it executes random actions with probability ϵ. Notably, our ϵ decays with the passage of time. In this research, we evaluate four different DRL approaches:

Deep Q-Network (DQN) Deep Q-Network is a model-free DRL approach that maximizes the Bellman equation:

$$Q_t(s_t, a_t) = Q_t(s_t, a_t) + \alpha(R_t(s_t, a_t) + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t)) \quad (1)$$

where, $Q_t(s_t, a_t)$ is the approximate Q-value (i.e., expected accumulated reward), $R_t(s_t, a_t)$ is the reward (r_t) at t, and α is the learning rate. At each t, the agent executes an action and stores the (s_t, a_t, r_t, s_{t+1}) in a replay buffer. After every n steps, the agent calculates the loss using the Eqn. 1 for random batches from the buffer. Here, we use a neural network as a universal function approximator of the Q-function $Q_t(s_t, a_t)$ (Riedmiller 2005; Sutton and Barto 2018). Consequently, the neural network parameters are updated by minimizing the loss in Eqn. 2 via stochastic gradient descent. Extensions include the Double DQN method that predicts $Q_t(s_{t+1}, a)$ and $Q_t(s_t, a)$ with two different neural networks for more stable Q-function updates.

$$Loss = R_t(s_t, a_t) + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t). \quad (2)$$

Actor-Critic Actor-critic approaches combine both the policy and value iteration methods using the following components:

- *Critic* is responsible for policy evaluation and uses a deep neural network (DNN) to estimate the Q-value. Based on the loss in Eqn. 2, the critic updates the parameters of the DNN and sends the computed gradients to the actor.
- *Actor* recommends the optimal action for the current state of a critic using a DNN, whose parameters are updated based on gradients received from the critic. Specifically, the actor searches for the optimal parameters of the DNN, θ^* (i.e., weights of DNN), that maximize the expected accumulated reward (Sutton and Barto 2018):

$$J(\pi_\theta) = E_{\pi_\theta}[\sum_{t=1}^{T} G(s_t, a_t)] \quad (3)$$

where, T is the number of decision epochs, and $G(s_t, a_t)$ is the total accumulated reward. The DNN parameters are updated using the following equation:

$$\theta = \theta + \alpha \nabla J(\theta) \quad (4)$$

where, α is the learning rate. In Eqn. 4, $\nabla J(\theta)$ is the gradient descent derived as follows (Sutton and Barto 2018):

$$\begin{aligned} \nabla J(\pi_\theta) &= \nabla E_{\pi_\theta}[\sum_{t=1}^{T} G(s_t, a_t)] \\ &= \nabla E_{\pi_\theta}(\sum_{t=1}^{T} \nabla_\theta log \pi_\theta(a_t|s_t) G(s_t, a_t)) \end{aligned} \quad (5)$$

where, $\pi(a_t|s_t)$ is the probability of taking action a_t in state s_t subject to the current policy parameters θ. We implement three different variants of actor-critic algorithms described as follows.

  - *Advantage Actor-Critic Approach (A2C)* This method uses an advantage function, $A(s_t, a_t)$, defined as

$$A_(s_t, a_t) = G(s_t, a_t) - V(s_t) \quad (6)$$

    to replace the $G(s_t, a_t)$ of Eqn. 5. The advantage function reduces the high variance to make the policy network more stable. Note that $V(s_t)$ in Eqn. 6 is the baseline value achieved at s_t.
  - *Asynchronous Advantage Actor Critic Approach (A3C):* A3C is different from A2C in that the actor updates the policy parameters asynchronously as soon as any gradient update from any critic (and does not wait for all critics to finish).

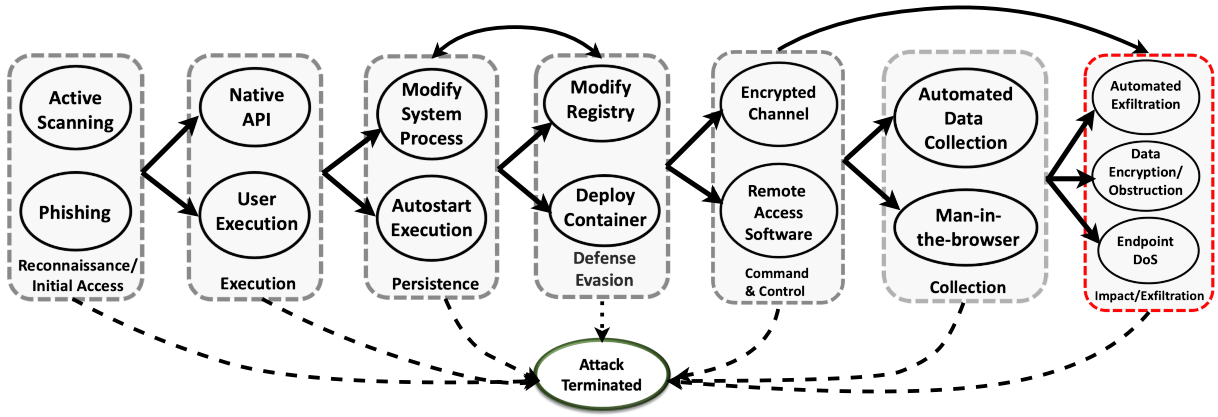


Figure 1: Multi-stage attack propagation represented with MITRE ATT&CK Tactics and Techniques. (Note: A directed edge between an attack tactic and technique specifies that the attacker may try to implement that technique next after achieving the objective of the attack tactic. Bidirectional arrow represents that *Defense Evasion* can come before *Persistence*.)

– *Proximal Policy Optimization (PPO)* This method clips the divergence of new policy when it is out of the region $[1-e, 1+e]$ (e is called clip parameter), in order to avoid drastic deviation from an older evaluated policy. This may be beneficial against sensor noises and errors. PPO optimizes the following clipped surrogate objective function, $L^{CLIP}(\theta)$:

$$L_t^{CLIP}(\theta) = \mathbb{E}[\min\{r_t(\theta)A_t, \text{clip}\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)}, 1-e, 1+e\right)A_t\}] \quad (7)$$

where, π_θ and π_{old} represents new and old stochastic policies respectively.

Autonomous Cyber Defense Framework

Figure 2 presents our proposed autonomous cyber defense framework. The core element of our framework is a custom OpenAI Gym (Brockman et al. 2016) simulation environment that we developed, where at each time-sequence, a DRL defense agent executes a *Defense Action* and observes *Attack Position* and *Reward* as feedback. Each episode (i.e., multiple time-sequences) considers an attack path (per Figure 1), where the red node in figure 2 represents the current *Attack Position*. Next, we describe adversary and defense models in our framework.

Adversary Model The objective of an adversary is to move from the *Reconnaissance/Initial Access* tactic to the goal tactic of *Impact/Exfiltration* sequentially (per Fig. 1). The adversary uses a logic-based model to achieve the objective of a specific tactic if they successfully execute one of their techniques. For example, from attack initiation an adversary may reach the *Reconnaissance/Initial Access* tactic by successfully executing either *Active Scanning* (e.g., scans to find vulnerable machines or services) or *Phishing* (e.g., sends malicious links to users). At any time, the attack position is the last successfully executed attack technique. The attack graph in Fig. 1 satisfies the monotonicity property, which means that an adversary never discards a specific attack position once they successfully execute it.

There are many possible chains/paths to move from *Reconnaissance/Initial Access* position to *Impact/Exfiltration*, from which, the attacker can choose any path based on the attack strategy. We assume that an adversary can change their strategy according to observations of previous attack impacts and network/system conditions. As a result, they can evade or defeat any static defense policy by discovering the deployed countermeasures.

To execute a technique successfully, the adversary needs to exploit at least one of the existing vulnerabilities that are still not discovered or cannot be patched due to usability, safety, or stability requirements. To do so, the adversary may follow diverse procedures based on the following logic. An attack attempt fails if they cannot exploit any relevant vulnerability due to countermeasures or lack of expertise. From any tactic/technique, the adversary moves to *Attack Terminated* position if their multiple attempts to ex-

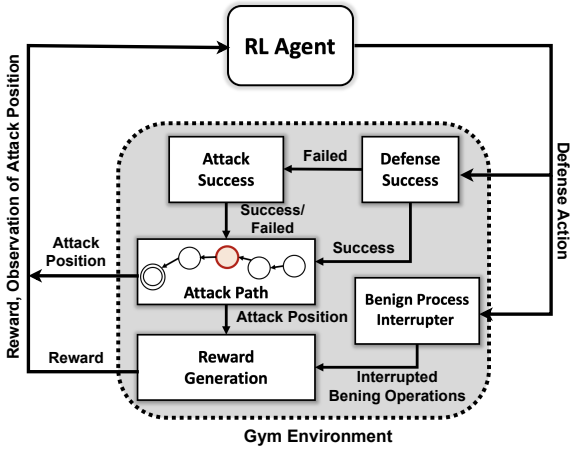


Figure 2: Autonomus cyber defense framework.

ecute next attack technique fails due to successful defense actions. Hence, there are two attack consequences: (i) adversary wins if they can successfully execute any technique under the *Impact/Exfiltration* position; (ii) adversary loses if they move to the *Attack Terminated* position. It is important to note that the adversary's strategy is implicitly captured in the description of the system's state evolution.

Defense Model The main objective of the defender (DRL agent) is to proactively prevent the adversary to reach the Impact/Exfiltration tactic phase, while minimizing loss due to interrupting benign operations. To determine the optimal defense, the agent needs to infer the current attack position and predict the next attack action. However, lack of domain-specific information on system complexities and adversarial behavior results in the defender facing the following uncertainties that prevents the creation of an apriori system dynamics model:

- *Uncertain Next Attack Technique:* The defense agent cannot predict the next attack technique due to two reasons. First, the defense agent has no prior knowledge of the attack graph in Fig. 1 as it requires domain-specific real-world attack sequences that are hard to obtain. Second, an adversary does not always follow the same strategy (attack sequence). For example, after *User Execution*, an adversary can either try to *Modify Registry* or *Modify System Process*.
- *Uncertain Next Attack Procedure:* The defense agent does not know the next attack procedure (i.e., attack actions required to implement an attack technique) or its likelihood due to lack of domain data.
- *Imperfect and Incomplete Observations:* Deployed alert mechanisms may not prove the current position of the adversary due to two reasons: (1) limited observability of processes, and (2) uncertain mapping from observations to attack techniques.

For our DRL experimentation, we assume that an adversary is at an initial access position at the start of each episode, and the defender leverages alert systems that monitor API calls to understand the current attack position. As mentioned before, this alert information is not only incomplete due to not observing all API calls but also imperfect due to probability of errors in mapping aggregated API calls to MITRE tactic and technique. Moreover, the defender only partially knows attack position but does not know which process is malicious, and therefore, seeks to learn the adversary's dynamic attack strategy governed by attack action sequences.

Defense Optimization Problem

We formulate the cyber defender's optimization problem using a Sequential Decision Process (SDP) (Braziunas 2003) construct, where the next attack technique depends on the current attack position and defense action. Note here that the adversary's optimal strategy is not learned in this work; instead, we train the defense agent against different adversarial strategies corresponding to distinct attack behaviors. Our SDP model is a tuple with four parameters: (S, A, R, γ), where S is the cyber system state space, A is the defense action space, R is the reward function, and γ is the discount factor. The agent only knows about S, A, and γ which define its interaction with the environment. Though the agent is unaware of the mathematical form of R. The objective in the SDP model is to compute an optimal action, $a^* \in A$, for any current state, $s \in S$ that maximizes the cumulative reward over a finite attack horizon.

At the start of each time t, the agent executes a defense action, $a_t \in A$, based on current environment state, $s_t \in S$, and receives feedback from the environment. Such feedback consists of current attack position, $s_{t+1} \in S$, as observation and defense payoff, r_t, as a reward. The payoff of a_t depends on its effectiveness on whether a_t prevented the next attack technique or not. To clarify, a_t is an effective defense action at s_t if it forces the attacker to stay at the previous position ($s_{t+1} = s_t$) or move to *Attack Terminated* node. Thus, this model also implicitly integrates the expected attack behavior into decision-model through learning effectiveness of defense actions at particular environment condition. Next, we describe the optimization model elements in detail.

State Space (S) Our state space, S, consists of 17 states, where each state, $s \in S$, is a sparse vector that represents an unique attack position. Hence, the current state at any time t specifies the position of adversary at t, based on which, the defense agent aims to choose the optimal defense action for that time-sequence. We consider three types of states: (1) 15 attack-technique states corresponding to each attack technique in Fig. 1; (2) *Attack Initiated* state, and (3) *Attack Terminated* state. The initial state is the *Attack Initiated* state, and the adversary moves to *Attack Terminated* state if they abort the attack due to failure or detection. Moreover, the adversary moves to a new state by successfully executing the associated attack technique. We consider two types of goal states:

- *Attack Goal State:* Adversary wins if they reach (1) *Automated Exfiltration*, (2) *Data Encryption/Obstruction*, or (3) *Endpoint Denial of Service (DoS)* state of *Impact/Exfiltration* tactic;
- *Defense Goal State:* Defender wins if the adversary moves to *Attack Terminated* state.

Defense Action Space (A) The defender's action space, A, has three different modes of operation:

1. *Inactive*: The defender remains silent and does nothing;
2. *Reactive*: The defender removes all processes that called or executed actions related to current attack position or last attack action;
3. *Proactive*: The defender blocks a specific set of API calls or operations to prevent the next attack action.

The defender remains *Inactive* for strategic reasoning or to avoid termination of critical benign processes. Whereas, for both *reactive* and *proactive* defense approaches, benign operations may be interrupted. We consider only one reactive action assuming that it can remove all processes associated with last attack action (whether successful or not). For proactive defense, we consider 21 distinct defense actions

that block unique sets of API calls and operations or adopt particular methods/measurements to prevent specific attack techniques. For instance, a proactive defense action: Restrict Registry Permission blocks the ability to change certain keys to prevent adversary's autostart execution (*Persistence* tactic) or registry modification (*Defense Evasion* tactic). Another proactive action such as Restrict File and Directory Permission (only certain set of sensitive files) can also stop autostart execution. However, success likelihood of different defense actions vary in both defense effectiveness (i.e., preventing attack action) and expected false positive rate (i.e., terminating benign operations). Hence, the defense agent must choose the best action considering defense effectiveness and expected false positive rate. Moreover, this research also assumes that the defender can execute one action at a time-sequence. Hence, for proactive defense approach, the defender's decision-model needs to understand what the adversary may do next to execute an effective mitigation action. The defender wins if the adversary moves to *Attack Terminated* position due to failure to execute new attack techniques or being detected and removed.

Reward Function (R) We consider the following reward function for the defense agent:

$$R = -p_g(s) \times I_g - \mathcal{I}_v \times I_g - C_f \tag{8}$$

where, $\mathcal{I}_v = -1$ if the defender wins, and 0 otherwise. Note that $p_g(s)$ is the probability of the adversary reaching the *Impact/Exfiltration* tactic from state s, and I_g is the impact/loss of a successful attack execution. Equation 8 comprise of three terms: (1) $p_g(s) \times I_g$ quantifies the risk at state s due to the probability ($p_g(s)$) of attacker's reaching to goal; (2) $I_v \times I_g$ quantifies the penalty or incentive to the defense agent when the adversary wins or loses, respectively; and (3) C_f is the cost of executing any defense action in A. The cost C_f depends on the aggregated loss due to interrupting benign operations, and defense implementation or operational cost. We assume that the defense implementation cost is same for all mitigation actions and zero cost for *Inactive* action.

Experiments

To implement our framework and solve the cyber defense optimization problem, an experimental plan was established to evaluate the performance of four DRL approaches (i.e., DQN, A2C, A3C, and PPO). The experimental setup, training and testing scenarios, adversary types, and simulation environment are described next.

Experimental Setup

We designed our experiments using Python 3.7 (Van Rossum and Drake Jr 1995), and used *RLlib* library (Liang et al. 2018) for implementing DRL algorithms. We have simulated our experiments using a Dell Alienware machine with 16-core 3GHz Intel Core i7-5960X processor, 64GB RAM, and three 4GB NVIDIA GM200 graphics cards. Although MITRE ATT&CK framework contains 11 tactics and many more techniques, we consider 7 tactics and 15 techniques for our experiments. Our defense action space A consists of 23 mitigation actions, including 21 proactive actions. Table 1 presents learning parameters that we used for all our experiments.

Parameter Name	Value
Entropy Coefficient	0.05
Initial exploration probability	1.0
Final exploration probability	0.04
Exploration delay period	300,000
Number of workers	4
Rollout fragment length	12
Batch size	48
PPO clip value	0.4
Training epochs	100
Steps per training epoch	25000

Table 1: Parameters used for training and testing simulation experiments. (Note: *Number of workers* refers to the number of parallel processes, and *Exploration delay period* refers to the number of training steps that decays exploration probability from 1.0 to 0.04.)

Training and Testing Scenarios

We generated all possible distinct attack paths from initial attack position (i.e., initial attack state) to any state of the last attack tactic (i.e., *Impact/Exfiltration* tactic). We used 80% attack paths for training and 20% attack paths for testing, where each attack propagation path is a unique sequence of attack techniques executed by an adversary to achieve their objective.

During training, each unique episode contains one attack propagation path, which ends if the defender/adversary wins or loses. The adversary wins or the defender loses in an episode if the adversary successfully executes all attack techniques across the attack path to satisfy *Impact/Exfiltration* tactic. However, an adversary loses or defender wins if they move to *Attack Terminated* state due to failing at least a number of times, n, across a path.

Adversary Types

For our experiments, we determine adversary types based on two attack parameters: (1) *skill*, and (2) *persistence*. With greater attack skill, success rate in executing next attack technique is higher due to increased capability in exploiting vulnerabilities. We define ρ as an attack skill parameter, representing attack success rate in exploiting a vulnerability. We assume that all vulnerabilities have same exploitability (i.e., impact and complexity). On the other hand, greater attack persistence indicates that the adversary does not abort their objective or cannot be readily detected despite their failed attempts. We define τ as an attack persistence parameter, representing the number of failed attempts before moving to attack terminated state (i.e., defense win). By tuning ρ and τ, we consider three different attack profiles/strategy: (1) Attack profile 1 (Av_1): $\rho = 0.75$ and $\tau = 4$, (2) Attack profile 2 (Av_2): $\rho = 0.85$ and $\tau = 5$, and (3) Attack profile 3 (Av_3): $\rho = 0.95$ and $\tau = 7$. For example, $Av_2(\rho = 0.85, \tau = 5)$ indicates that the adversary with profile 2 has 85% success likelihood in exploiting a specific vulnerability and tolerates

maximum 5 failed attempts in an episode. Thus, attack profile 3 represents the most sophisticated adversary, and attack profile 1 represents a naïve adversary. It is important to note that the adversary may change attack procedures if their previous attempt in exploiting a specific attack technique fails.

Simulation Environment

We developed a custom OpenAI Gym simulation environment (not in the public domain at this time) for the autonomous cyber defense framework illustrated in Figure 2. The DRL defense agent has no knowledge of the environment and attack behavior. The defense agent receives observations about next attack position and reward as feedback. The agent then determines the current attack position based on recent observation. We assume that system alerts can be translated to a specific attack position with 85%, 75%, and 65% accuracy against Av_1, Av_2, and Av_3, which indicates that the adversary is more stealthy with increased level of sophistication.

Within the simulation environment, at the start of each episode, an attack propagation path is selected, and the environment is set to the initial attack position. At the start of each timestep, a *Defense Success* module determines whether the current defense action stopped the current attack technique or not, based on the correlations among attack procedures and defense actions. Another module, *Benign Process Interrupter*, sends the list of benign processes/-operations interrupted due to the recent defense action to the *Reward Generation* module. We assume a power-law distribution function to generate the number of interrupted benign operations, which gradually becomes lower towards the last attack tactic. If the defense action is successful, the adversary stays at their current position. Otherwise, if the defense action fails, *Attack Success* determines the success of the current attack technique based on attack skill (ρ) of attack profile. If the attack is successful, the adversary moves to next position; otherwise, they remain at the current position or move to *Attack Terminated state* (end of episode). The reward is generated based on attack position, interrupted benign operations, and others using Eqn. 8. At the end of the timestep, reward and next observation are sent to the DRL defense agent that is used to update and refine their policy.

Evaluation

We assess the performance of different DRL algorithms using the *Defense-Win Ratio* (DWR) metric, which evaluates the fraction of episodes where the defender won in a single batch, where each batch comprises of 200 episodes. Using the DWR ratio, we analyze (i) the convergence of the DRL algorithms during training, and (ii) the performance of the trained defense models against unseen attack sequences during testing. We evaluate all four DRL algorithms against three attack profiles (as described in Section).

Hyperparameter Optimization and Training

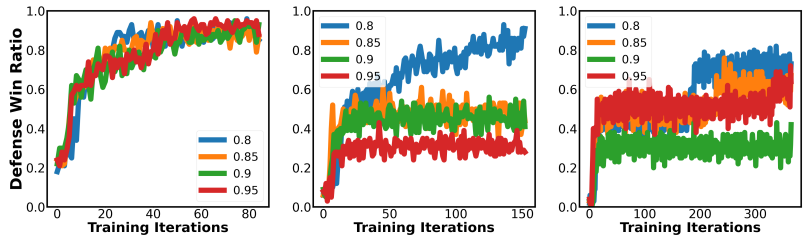
For brevity, here we only illustrate the training performance of DQN and A2C using the DWR metric by varying two hyper-parameters: discount factor (γ) and learning rate (α). In the following sections, each figure has two rows and three columns; the columns refer to the three attack profiles, while the rows correspond to the hyperparameters (the first row is for γ and the second row is for α). In this research, for all DRL algorithms, we have used fully connected neural network with 2 hidden layers where each of them has 256 neurons. We have used *tanh* as activation function in all the cases.

A2C Figure 3a illustrates the sensitivity of A2C to different values of γ, keeping α fixed to 0.005. Against attack profile 1, all A2C instances converge at DWR value of 0.95 within 40 iterations (40×200 episodes). Against attack profile 2, the A2C instance with $\gamma = 0.8$ achieves the highest DWR ratio, while all other discount factors induce poor performance. For attack profile 3, $\gamma = 0.8$ reaches the highest DWR of 0.8 while requiring 200 iterations. Thus, attack sophistication not only increases the convergence time but also reduces the defense agent's success rate. Averaging across all attack profiles, A2C performs best with $\gamma = 0.8$, which shows that the defense agent must balance the trade-off between the current reward and possible future payoff. Fig. 3b exhibits the sensitivity of A2C (with optimal γ set to 0.8) for different values of α. We observed that the algorithm performance does not change significantly against attack profile 1. Interestingly, against both attack profiles 2 and 3, A2C with $\alpha = 0.0005$ shows the best performance; this is possibly because the algorithms get stuck to local minima for larger values of α. Therefore, for A2C we set $\alpha = 0.005$ and $\gamma = 0.8$ for the test experiments.

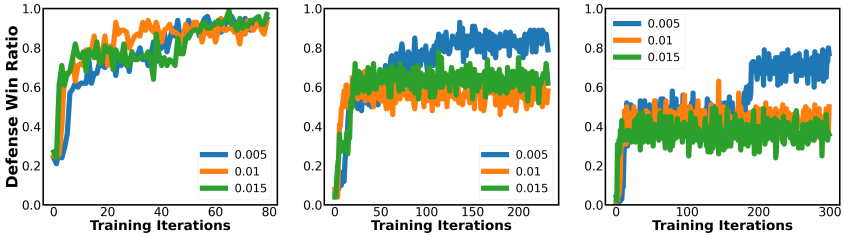
DQN We followed the same training approach for DQN as that of A2C. From Fig. 4a, we observe that DQN converges to optimal policy within 10 iterations, which is much faster compared to A2C; in fact, it was better than all of the DRL algorithms we tested. Moreover, DWR was higher than other DRL methods for each attack profile, which showed the superior performance of DQN for environments with discrete states and actions. Moreover, Fig. 4b illustrated that DQN-performance does not change significantly with change in α. For testing, we set $\gamma = 0.8$ and $\alpha = 0.01$ for DQN.

Testing Results

We discuss the testing performance of the DRL algorithms against the three attack profiles with unseen attack sequences not used during training. Fig. 5 illustrates the cumulative result in defending against the adversary at different phases. Here, each phase corresponds to a distinct adversary tactic. *Tactic ID: 0* specifies the initial attack position before *Reconnaissance*, *Tactic ID: 6* specifies the *Collection*, and all other tactics in Fig. 1 are numbered sequentially from *Tactic ID: 1* to *Tactic ID: 5*. Note that *Impact/Exfiltration* (tactic ID 7) is not shown in Fig. 5 as the defender loses if the attacker reaches that state. Table 2 reports how many attack sequences were stopped at the corresponding attack tactic. For example, the column corresponding to *Tactic ID: 3* specifies how many attacks were stopped at *Defense Evasion*; similarly for the other columns. The *Mean Reward* column specifies how much reward is achieved compared to the best reward (i.e., when stopping all attacks at Tactic ID: 0).

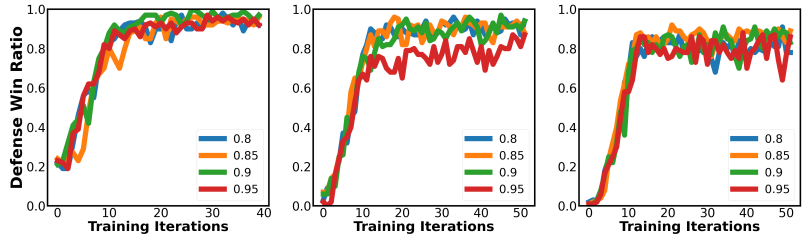


(a) Convergence for different values of γ (α set to 0.005).

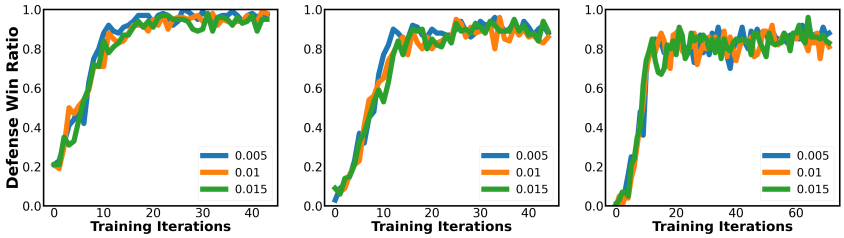


(b) Convergence for different values of α (with γ set to optimized value of 0.8).

Figure 3: Sensitivity of A2C for different values of γ and α against attack profiles Av_1, Av_2, and Av_3 (left to right).



(a) Convergence for different values of γ (α set to 0.005).



(b) Convergence for different values of α (with γ set to optimized value of 0.8).

Figure 4: Sensitivity of DQN for different values of γ and α against attack profiles Av_1, Av_2, and Av_3 (left to right)

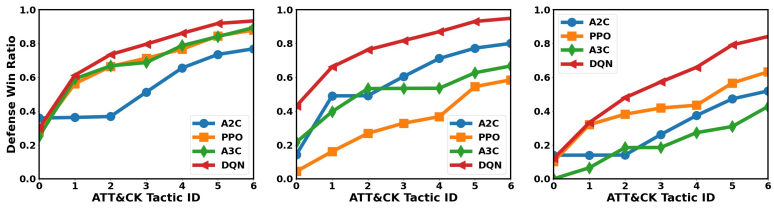


Figure 5: DWR performance during testing against the three attack profiles Av_1, Av_2, and Av_3 (from left to right).

The objective of the defense DRL agent is not only to stop the adversary from moving to *Impact/Exfiltration* state but also to stop the attack progression as early as possible. Against all attack profiles, we observed that DQN exhibited the best performance in stopping the adversary as soon as possible. As can be seen in Table 2 and Fig. 5, DQN stops the three attack profiles within the *Defense Evasion* (tactic ID 3) in 79.6%, 82%, and 57.3% of the cases respectively, while achieving corresponding defense success rates of 93.3%, 95%, and 84.1%. Against Av_3, DQN has a lower

DRL Algorithm	Attack Profile 1 (in %)			Attack Profile 2 (in %)			Attack Profile 3 (in %)		
	Tactic ID: 3	Tactic ID: 6	Mean Reward	Tactic ID: 3	Tactic ID: 6	Mean Reward	Tactic ID: 3	Tactic ID: 6	Mean Reward
A2C	51.3	76.9	56.7	60.5	80	59.8	26.2	51.9	27.11
PPO	71.3	87.9	70.3	32.9	58.5	36.8	41.9	63.2	40
A3C	68.8	89.4	72	53.5	66.7	48	18.5	42.8	18.5
DQN	79.6	93.3	77	82	95	77.2	57.3	84.1	60.7

Table 2: Testing performance comparison among different DRL algorithms.

success rate than other attack profiles, as the sophisticated adversary hardly fails in exploiting a vulnerability. Besides, Av_3 is persistent and does not give up easily in spite of failed attempts, which leads to lower DWR values for the defense agent. Note that other DRL algorithms did not consistently perform well against all three attack profiles. In fact, the next-best performing algorithm against Av_3 had approximately 50% success rate. This is possibly due to the fact that actor-critic methods typically require more training samples than DQN to show better test accuracy.

Conclusion

Application of DRL methods for cyber system defense are promising, especially under dynamic adversarial uncertainties and limited system state information. Evaluating multiple DRL algorithms trained under diverse adversarial settings is an important step toward practical autonomous cyber defense solutions. Our experiments suggest that model-free DRL algorithms can be effectively trained under multi-stage attack profiles with different skill and persistence levels, yielding favorable defense outcomes in contested settings. However, some practical challenges that need to be addressed further in using model-free DRL include (Dulac-Arnold, Mankowitz, and Hester 2019): (i) explainability of the black-box DRL policies, (ii) vulnerability to adversarial noise and data poisoning, and (iii) convergence for large state-action spaces. Future work will include developing DRL-based transfer learning approaches within dynamic environments for distributed multi-agent defense systems.

Acknowledgments

This research was supported by the U.S. Department of Energy, through the Office of Advanced Scientific Computing Research's "Data-Driven Decision Control for Complex Systems (DnC2S)" project. Part of this research was supported by the Mathematics for Artificial Reasoning in Science (MARS) initiative at Pacific Northwest National Laboratory (PNNL) under the Laboratory Directed Research and Development (LDRD) program. PNNL is multiprogram national laboratory operated by Battelle for the U.S. Department of Energy under contract DE-AC05-76RL01830.

References

Braziunas, D. 2003. POMDP solution methods. https://www.techfak.uni-bielefeld.de/~skopp/Lehre/STdKI_SS10/POMDP_solution.pdf. (Date last accessed 09-November-2022).

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. OpenAI Gym. *arXiv preprint arXiv:1606.01540*.

Carin, L.; Cybenko, G.; and Hughes, J. 2008. Cybersecurity strategies: The queries methodology. *Computer*, 41(8): 20–26.

Caulfield, T.; and Fielder, A. 2015. Optimizing time allocation for network defence. *Journal of Cybersecurity*, 1(1): 37–51.

Chatterjee, S.; Halappanavar, M.; Tipireddy, R.; and Oster, M. 2016. Game theory and uncertainty quantification for cyber defense applications. *SIAM News*, 49(6).

Chen, D.; Xu, M.; and Shi, W. 2018. Defending a cyber system with early warning mechanism. *Reliability Engineering & System Safety*, 169: 224–234.

Dulac-Arnold, G.; Mankowitz, D.; and Hester, T. 2019. Challenges of Real-World Reinforcement Learning. In *Proceedings of 36th International Conference on Machine Learning (ICML)*. arXiv:1904.12901.

Dutta, A.; Al-Shaer, E.; and Chatterjee, S. 2021. Constraints Satisfiability Driven Reinforcement Learning for Autonomous Cyber Defense. In *Proceedings of 1st International Conference on Autonomous Intelligent Cyber-Defence Agents (AICA)*. arXiv:2104.08994.

Fovino, I. N.; Masera, M.; and De Cian, A. 2009. Integrating cyber attacks within fault trees. *Reliability Engineering & System Safety*, 94(9): 1394–1402.

Frigault, M.; Wang, L.; Singhal, A.; and Jajodia, S. 2008. Measuring network security using dynamic bayesian network. In *Proceedings of the 4th ACM workshop on Quality of protection*, 23–30.

Hu, X.; Xu, M.; Xu, S.; and Zhao, P. 2017. Multiple cyber attacks against a target with observation errors and dependent outcomes: Characterization and optimization. *Reliability Engineering & System Safety*, 159: 119–133.

Islam, M. M.; Dutta, A.; Sajid, M. S. I.; Al-Shaer, E.; Wei, J.; and Farhang, S. 2021. CHIMERA: Autonomous Planning and Orchestration for Malware Deception. In *2021 IEEE Conference on Communications and Network Security (CNS)*, 173–181. IEEE.

Liang, E.; Liaw, R.; Nishihara, R.; Moritz, P.; Fox, R.; Goldberg, K.; Gonzalez, J.; Jordan, M.; and Stoica, I. 2018. RLlib: Abstractions for distributed reinforcement learning.

In *International Conference on Machine Learning*, 3053–3062. PMLR.

Liang, X.; and Xiao, Y. 2012. Game theory for network security. *IEEE Communications Surveys & Tutorials*, 15(1): 472–486.

MacDermed, L.; Isbell, C.; and Weiss, L. 2011. Markov games of incomplete information for multi-agent reinforcement learning. In *Workshops at the Twenty-Fifth AAAI Conference on Artificial Intelligence*.

McCarthy, S. M.; Sinha, A.; Tambe, M.; and Manadhata, P. 2016. Data exfiltration detection and prevention: Virtually distributed pomdps for practically safer networks. In *International Conference on Decision and Game Theory for Security*, 39–61. Springer.

MITRE. 2022. MITRE ATT&CK. https://attack.mitre.org/. (Date last accessed 09-November-2022).

Nguyen, T. T.; and Reddi, V. J. 2021. Deep Reinforcement Learning for Cyber Security. *IEEE Transactions on Neural Networks and Learning Systems*, 1–17.

Poolsappasit, N.; Dewri, R.; and Ray, I. 2011. Dynamic security risk management using bayesian attack graphs. *IEEE Transactions on Dependable and Secure Computing*, 9(1): 61–74.

Ramuhalli, P.; Halappanavar, M.; Coble, J.; and Dixit, M. 2013. Towards a theory of autonomous reconstitution of compromised cyber-systems. In *2013 IEEE International Conference on Technologies for Homeland Security (HST)*, 577–583. IEEE.

Riedmiller, M. 2005. Neural fitted Q iteration–First experiences with a data efficient neural reinforcement learning method. In *Proceedings of the 16th European Conference on Machine Learning (ECML), Porto, Portugal, October 3-7, 2005. Proceedings 16*, 317–328. Springer.

Roy, S.; Ellis, C.; Shiva, S.; Dasgupta, D.; Shandilya, V.; and Wu, Q. 2010. A survey of game theory as applied to network security. In *2010 43rd Hawaii International Conference on System Sciences*, 1–10. IEEE.

Sarraute, C.; Buffet, O.; and Hoffmann, J. 2012. POMDPs make better hackers: Accounting for uncertainty in penetration testing. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*.

Shin, J.; Son, H.; Heo, G.; et al. 2015. Development of a cyber security risk model using Bayesian networks. *Reliability Engineering & System Safety*, 134: 208–217.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement Learning: An Introduction*. MIT press.

Tipireddy, R.; Chatterjee, S.; Paulson, P.; Oster, M.; and Halappanavar, M. 2017. Agent-centric approach for cybersecurity decision-support with partial observability. In *2017 IEEE International Symposium on Technologies for Homeland Security (HST)*, 1–6. IEEE.

Van Rossum, G.; and Drake Jr, F. L. 1995. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam.