Knowledge Guided Two-player Reinforcement Learning for Cyber Attacks and Defenses

Aritran Piplai*, Mike Anoruo*, Kayode Fasaye*, Anupam Joshi*, Tim Finin*, Ahmad Ridley†
*Dept. of Computer Science & Electrical Engineering, University of Maryland, Baltimore County,
Email: {apiplai1, manoruo1, kfasaye1, joshi, finin}@umbc.edu
† Laboratory for Advanced Cybersecurity Research, National Security Agency
Email: adridle@uwe.nsa.gov

***Abstract*—Cyber defense exercises are an important avenue to understand the technical capacity of organizations when faced with cyber-threats. Information derived from these exercises often leads to finding unseen methods to exploit vulnerabilities in an organization. These often lead to better defense mechanisms that can counter previously unknown exploits. With recent developments in cyber battle simulation platforms, we can generate a defense exercise environment and train reinforcement learning (RL) based autonomous agents to attack the system described by the simulated environment. In this paper, we describe a two-player game-based RL environment that simultaneously improves the performance of both the attacker and defender agents. We further accelerate the convergence of the RL agents by guiding them with expert knowledge from Cybersecurity Knowledge Graphs on attack and mitigation steps. We have implemented and integrated our proposed approaches into the CyberBattleSim system.**

I. Introduction

Signature-based cybersecurity defense mechanisms underperform when faced with new challenges from sophisticated adversaries[1]. Reliance on attack signatures and Indicators of Compromise alone are not enough to mitigate a significant number of current cyber-threats, especially those from state actors or large transnational cybercriminal enterprises. For instance, 99% of malware programs are used only once in their current form before being modified for reuse [1]. Such modifications can render signature-based defenses useless. This a major reason why we see record numbers of zero-days in recent years [2]. Another significant drawback of signature-based methods is their inability to combat multi-vector cyber attacks. For example, there can be multiple points of entry into a machine's network, either through vulnerabilities of the machine itself or via vulnerabilities of other machines that expose the first machine's credentials. In order to defend against these attacks, we need powerful simulation platforms to test and evaluate our solutions.

An improvement over traditional rule-based or signature-based detection techniques are systems that use supervised machine learning (ML) models. However, they suffer from many of the same shortcomings, since they are typically trained on historical data and have limited ability to generalize. This often makes them unable to detect zero days. We need an improvement over supervised ML models to find more robust solutions for unknown and multi-vector attacks [3]. Recently, SR Labs analyzed Endpoint Detection and Response (EDR) systems to find out that relying on specific 'triggers' to launch malware defense mechanisms may not be helpful [4]. This makes it very easy for skilled attackers to evade the system. Simulating multiple attack scenarios and testing defenses will lead to a more 'generic' defense against attackers. The authors state that generic defenses are more difficult to evade than defenses that work on 'triggers'. With the help of reinforcement learning, we simulate these attack scenarios to create defenses that are more generic and do not rely on a specific trigger.

Recent advancements in Reinforcement Learning (RL) have shown promising results for generating adversaries as well as mitigating them with defense mechanisms. RL algorithms can work particularly well for cybersecurity [5] because they can leverage their exploration capabilities to discover previously unknown attack and defense scenarios. Microsoft recently released an open-source cyber-battle simulation platform called 'CyberBattleSim' [6]. This helps practitioners generate simulated environments of cyber defense exercises (CDX) and capture-the-flag (CTF) scenarios.

In this simulation-based approach, there is an RL agent that tries to attack and take over a network, called the attacker agent. The RL agent aims to explore different types of exploits on vulnerabilities mentioned in the system. Although we have provisions for generating cyber-battle scenarios, it needs a significant amount of human effort to explicitly mention what the results of a successful exploit will be at a given machine, or node, in the network. An external knowledge source that captures semantic information about different machine types, vulnerabilities, and mitigation can greatly help in this regard. Providing such a knowledge source as knowledge graphs is one of our contributions in this paper.

CyberBattleSim also has naive defenders that take random defensive actions, completely unaware of the attackers' actions. The attacker becomes more intelligent, but we need to know how the attacker would perform under more sophisticated defense mechanisms. The number of steps to reach a particular goal can vary based on how well the agents explore their corresponding environments. An external knowledge source can also prove beneficial in this regard to guide exploration to states that are more likely to yield better results.

In this paper, we describe a knowledge-guided two-player RL algorithm for cyber defenses and attacks that makes the attacker more robust, while also greatly improving the performance of the defender. We test this algorithm in the CyberBattleSim environments under different scenarios. The key contributions of this research are as follows.

- **Knowledge generated CDX environments**: We use knowledge graphs to generate the environment for simulating CDXs. This greatly reduces the human effort required to explicitly describe each vulnerability and the effect of each exploit on them.
- **Sophisticated defender**: We use an RL-based defender as opposed to a random defender and simultaneously train the attacker and defender agents.
- **Knowledge guided exploration**: We use prior knowledge to bias the exploration towards actions that are likely to be more effective, reducing the time needed to reach a pre-specified goal for both the attacker and defender.

We organize our paper as follows. We cover some of the relevant work in the area in Section II. Section III describes the key components of our system in more detail. We present key research results in Section IV and conclude and discuss the next steps in Section V.

II. Related Work

In this section, we describe some of the relevant work in this research area.

A. Knowledge Graphs for Cybersecurity

CKGs are widely used to represent Cyber Threat Intelligence (CTI). There is a significant body of work that concentrates on extracting CTI from open-source text and representing them in CKGs. Usually, Natural Language Processing methods are used to retrieve information from open-source text such as Twitter feeds [7], Malware After Action Reports [8, 9], and others [10]. A large number of CKG schemas are based on STIX [11], an industry standard for exchanging threat intelligence. Other varieties of CKGs exist that have their own schema for representing CTI [12, 13]. CKGs have various applications for security analysts. They have been used to detect malware activity [14], and also for malware comparison [15]. A CKG that appropriately captures vulnerabilities for specific machines, attack patterns used to exploit these vulnerabilities, and the mitigation steps required to thwart the attack is useful in this paper. Such Knowledge graphs can also capture security-related policies[16, 17], and even trust between different entities in the system that provide knowledge[18, 19].

B. ML and RL in Cybersecurity

ML algorithms have been successful in the domain of cybersecurity, and there has been considerable research in this area. They have been particularly effective in Intrusion Detection Systems [20], malware detection [21], and detecting attacks in Cyber-Physical Systems [22]. However, with the recent spike in the number of zero days in the cybersecurity space, it has become more difficult for supervised machine learning algorithms to detect them. Highly accurate supervised AI models for Intrusion Detection systems are also susceptible to adversarial attacks [23]. In order to detect unseen attacks, there has been an increased interest in RL for cybersecurity in recent years. RL and Deep RL have been used in other domains, such as recommendation engines [24, 25] and simulating action games [26]. We also see applications of RL in wireless security to prevent jamming attacks [27]. Xu et al. [28] proposed a kernel-based RL approach using Least-Squares Temporal-Difference (LS-TD) for intrusion detection outperforming Hidden Markov Models. For autonomous cyber defense generation, RL has been proven to be effective by Ridley et al. [29]. There has also been an increased interest in identifying zero days with RL. For example, Hu et al. [30] proposed a method to use Attack Graphs for zero-day attack simulation with RL. Prior knowledge has been used in RL to find out appropriate reward functions for malware detection [14]. However, for most cyberbattle simulations the reward functions are straightforward. In our paper, we use a minimax DQN [31] along with CKGs to simulate attacks and defenses.

C. Cyber Battle Simulation

A large number of cybersecurity simulation environments currently exist for practitioners to understand cyber-attacks and be better prepared to prevent them. A cybersecurity simulation platform called ‘Insight’ has been proposed to understand zero-day attacks [32]. Another simulation platform called ‘DETERlab’ became very useful for simulating hosts and networks [33]. Recently other simulation platforms have also become popular, such as ‘GALAXY’ [34], ‘CANDLES’ [35], and ‘FARLAND’ [36]. ‘CANDLES’ has provisions for simultaneously training the defender agent and the attacker agent with genetic algorithms. Recently, another simulation platform called ‘CybORG’ [37] claimed to have addressed the shortcomings of other simulation platforms by adding emulation capabilities and provisions for training a defender. In our paper, we address the challenges of adding an adaptive defender, and we leverage the capabilities of CKGs to generate simulation environments easily.

III. Methodology

In this section, we describe the key components of our system. We have a CKG that represents semantic information about vulnerabilities and mitigation steps. This helps us generate the network graph in our simulation platform which in turn, creates the environment for the RL agents to run. Figure 2 describes the components of our system. Next, we discuss each component in detail.

A. Cybersecurity Knowledge Graphs

We use the schema of the CKGs that have been used to represent STIX data[8, 9, 38]. These CKGs are populated with information extracted from STIX [11] and contain semantic information about cyberattacks collected from multiple sources, such as TAXII servers as well as from the CVE [39], and CWE [40] datasets. Apart from the knowledge collected from these semi-structured sources, we use a deep learning-based CKG

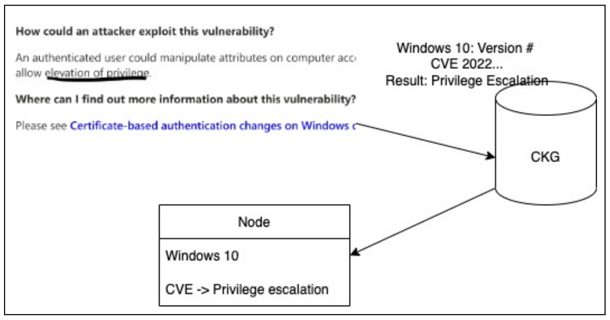

How could an attacker exploit this vulnerability?
An authenticated user could manipulate attributes on computer acc
allow elevation of privilege.
Where can I find out more information about this vulnerability?
Please see Certificate-based authentication changes on Windows c
Windows 10: Version #
CVE 2022...
Result: Privilege Escalation
CKG
Node
Windows 10
CVE -> Privilege escalation

Fig. 1: Feeds are parsed to identify vulnerabilities and to generate the network graph for the simulation.

construction pipeline [9] that collects data from unstructured text. We further improve this CKG by parsing additional text for vulnerabilities. We further enhance the knowledge in our CKG by creating a function that queries the National Institute of Standards and Technology (NIST) cybersecurity database to obtain a list of CVEs. This function takes in a list of operating systems and searches for known CVEs relating to a given Operating system using NIST's API. It compiles a list of vulnerabilities for each operating system, stores it within a dictionary, and returns it to the user for further use.

The entity classes are mapped to the classes in STIX [11], which is an industry standard for exchanging threat intelligence. Out of all the classes, we concentrate on four specific classes: *Exploit-Target*, *Attack-Pattern*, *Vulnerability*, and *Course-of-Action*.

Our *CyberBattleSim* simulation platform expects practitioners to define the network, specifying each machine or node. A few node properties are also required, along with vulnerabilities associated with the node and the result of a successful vulnerability exploit. The network graph also requires information about the credentials that can be leaked as a result of a successful exploit. The credentials can be of a particular node or a remote node that can be accessed from the current node.

We use the *Common Vulnerability Scoring System* (CVSS) [41] to determine the amount of control an attacker may get over a node if an exploit is successful. CVSS scores range from 1 to 10, with 10 being the highest access that can be guaranteed by exploiting the vulnerabilities. If the CVSS score is greater than nine, we assume that the attacker has total control over the node, and the system-level files can reveal a remote node's credentials. We use this as guidance. However, the agent is not limited by the vulnerabilities suggested by the CKG as there may not be enough vulnerabilities to exploit with a CVSS greater than 9. If a Linux node is accessed through 'SSH' from another node, say Node X, it is discovered by the attacker if a successful exploit of a vulnerability with a higher CVSS score has taken course on Node X. These pieces of information are parsed and asserted in the CKG that we use. Through simple queries, we can retrieve this information about the network's nodes. We write simple rules to populate the properties of the nodes that we define. For example, in Figure 1, we can see how information relating to a vulnerability 'CVE-2022-26923' present in a particular version of Windows 10 is parsed and asserted to the CKG. The same information is used to generate the vulnerability properties of a node in the network. It should be noted that these actions are suggested by the CKG to the RL algorithm, but it does not limit the exploration of the RL algorithm to these vulnerabilities.

B. RL Agents

The vanilla Deep Q-Network (DQN) [42] is based on a Markov Decision Process (MDP) of the form $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{T}$ is the transition matrix that tells us the next state $\mathcal{S}'$ from a given state $\mathcal{S}$, $\mathcal{R}$ is the rewards received, and γ is the discount factor. The Bellman equation for updating the Q values for a state action pair (s, a) is as follows.

$$\underbrace{Q(s,a)}_{\text{New Q-Value}} = Q(s,a) + \underbrace{\alpha}_{\text{Learning Rate}} [\underbrace{R(s,a)}_{\text{Reward}} + \underbrace{\gamma}_{\text{Discount rate}} \overbrace{\max Q'(s',a')}^{\substack{\text{Maximum predicted reward, given} \\ \text{new state and all possible actions}}} - Q(s,a)]$$

In the equation above, Q values for state-action pairs are updated with the help of the current Q values and the discounted future rewards of the next states. The DQN for the attacker agent is modeled based on the equation above. We have a policy network $Q(s)$ and a target network $Tar(s)$. The defender agent in CyberBattleSim performs actions randomly. However, random actions are not sophisticated enough to defend against an attacker agent being trained with the DQN. These actions do not observe the state space, and they do not have knowledge about the possible vulnerabilities that can be exploited by the attacker. In order to train the attacker and the defender jointly, we model a minimax-based two-player DQN [31].

The categories of actions for the attacker are as follows.

- Local attack
- Remote attack
- Connect

We use the following action categories for the defender.

- Patch vulnerability
- Re-image machine
- Add firewall rule
- Remove firewall rule
- Shut down service/port

For the attacker, the action space is $|Attack_{local}| + |Attack_{remote}| + |Connect| + 1$, where $|A|$ represents the cardinality. For defense, the action space is $|Vulnerability_{local}| + |Vulnerability_{remote}| + |Services| + |Nodes| + 2 + 1$. The actions of adding and removing firewall rules are random, so we need to add two at the end to represent those actions. In both cases, we add one that symbolizes no action at a particular step.

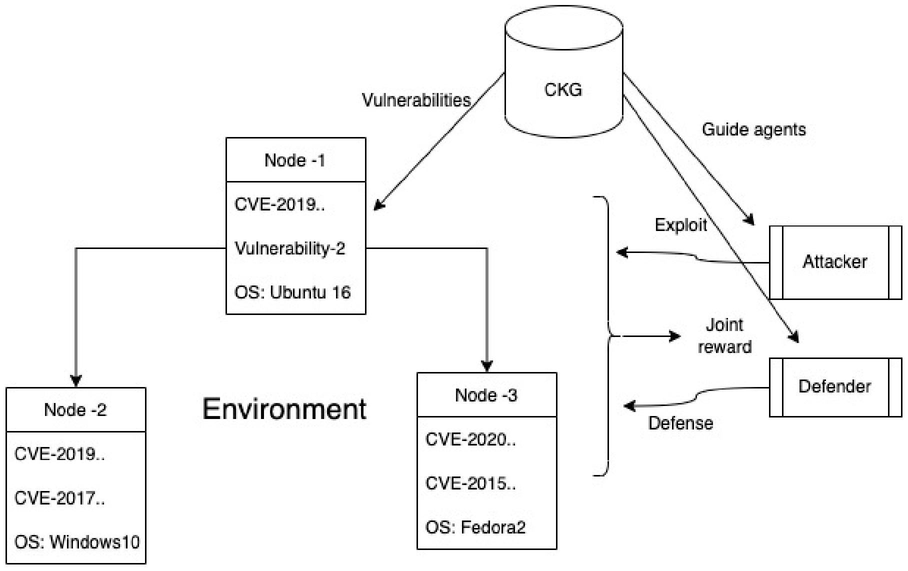


Fig. 2: An architecture diagram specifying the different components of our model. CKG has information about vulnerabilities of different nodes based on their operating systems and versions. This information is used to populate the node properties. This collection of nodes and properties forms the environment. The two agents, the attacker and defender, also receive guidance from the CKG. The joint reward is observed for both the attacker and defender from the environment after taking the actions 'Exploit' and 'Defense' respectively.

For a two-player minimax game the MDP is represented as $(\mathcal{S}, \mathcal{A}, \mathcal{B}, \mathcal{T}, \mathcal{R}, \gamma)$. The term $\mathcal{B}$ represents the defender action space. Compared to vanilla DQN, the state transition takes place as $S' = T(S, a_{attack}, b_{defend})$. The reward is observed jointly after observing the state S'. The goal of the attacker is to maximize the reward, and the goal of the defender is to minimize the reward. The two-player game is successfully able to learn two separate policies, one for the attacker π and one for the defender ϕ, jointly.

$$\begin{aligned} &Q(s, a_{attack}, b_{defend}) \leftarrow \\ &(1-\alpha) \cdot Q(s, a_{attack}, b_{defend}) + \alpha \cdot \{r(s, a_{attack}, b_{defend}) \\ &\quad + \gamma \cdot \max_{\pi' \in \mathcal{P}(\mathcal{A})} \min_{\phi' \in \mathcal{P}(\mathcal{B})} \mathbb{E}_{a'_{attack} \sim \pi', b'_{defend} \sim \phi'} \\ &\qquad [Q\left(s', a'_{attack}, b'_{defend}\right) \end{aligned} \tag{1}$$

In Equation 1, we see how Q-values are updated jointly, and two separate policies are updated. At a particular timestep, we sample attacker actions a_{attack} and defender actions b_{defend}, either through exploration or exploitation. We calculate the discounted future rewards by taking the min-max of the expected future rewards under the defender's current policy ϕ' and the attacker's current policy π'.

$$\begin{aligned} &Tar_i \leftarrow \{r(s, a_{attack}, b_{defend}) \\ &\quad + \gamma \cdot \max_{\pi' \in \mathcal{P}(\mathcal{A})} \min_{\phi' \in \mathcal{P}(\mathcal{B})} \mathbb{E}_{a'_{attack} \sim \pi', b'_{defend} \sim \phi'} \\ &\qquad [Q\left(s', a'_{attack}, b'_{defend}\right) \end{aligned} \tag{2}$$

The Minimax DQN has two neural networks: a policy network Q and a target network Tar. The target value is represented by Equation 2, whereas the policy network update is shown in Equation 3.

$$\widetilde{Q}_{k+1} \leftarrow \min \sum_{i=1}^{n} \left[Tar_i - Q\left(S_i, a_{attack}, b_{defend}\right)\right]^2. \tag{3}$$

The number of actions, in this case, becomes: $|Attacker_{actions}| \cdot |Defender_{actions}|$.

A sequence of features represents the state space. The feature set has two segments: global features and node-specific features. The global feature set indicates the attacker's current state and comprises counts of discovered nodes, discovered ports, and discovered credentials. Node-specific features include the success and failure counts of attempted exploits at each node. We also include three additional features. The first is an array of services that are active at each node. This is very important because the defender is able to shut down services, and the attacker should be aware of it through the features representing the state. The other features are also specific to defender actions, such as the number of vulnerabilities active in the state, and the number of firewall rules that allow traffic to each node. These additional features are included so that both the attacker and the defender are aware of each other's actions. The additional features manifest the actions of the defender to ensure that the attacker is still at an advantage.

C. Guiding Agents

DQN algorithms create an experiential replay buffer with a collection of sample points that are individual entries of the MDP defined in Section III-B. Each data point of the MDP $(\mathcal{S}, \mathcal{A}, \mathcal{B}, \mathcal{T}, \mathcal{R}, \gamma)$ is achieved through exploring attacker actions $a \in \mathcal{A}$ and defender actions $b \in \mathcal{B}$. We use the standard approach of ϵ-greedy action selection to generate our experiential replay buffer that forms our training set for training the DQN. In ϵ-greedy action selection, an agent explores more at the beginning of an episode and begins to exploit what it has learned more as the number of iterations increases. The defender can use information from our CKGs to guide its exploration of actions more favorable to defenders.

```
$C \leftarrow KG(Vuln = X, Node = N)$
**if** $C \neq \emptyset$ **then**
    $b*$ = $\min_b D[b - C]$
    Generate($P_{exp}(b)$)
**else**
    Generate uniform distribution
**end if**
Initialize($Q$, $S$)
**while** $done \neq True$ **do**
    **if** $x < \epsilon$ **then**
        $b* \leftarrow Sample(P_{exp}(b))$
        $Shuffle(b) \forall b \neq b*$
        $b \leftarrow b*$
        $Dilate(P_{exp})$
        $a \leftarrow Explore(a)$
    **else**
        $b$ = $\min_b \max_a (Q(s, a, b))$
        $a$ = $\max_a \min_b (Q(s, a, b))$
    **end if**
    $S', R \leftarrow (S, a, b)$
    $Tar_i = R + \gamma * \max_{a'} \min_{b'} [Q(S', a', b')]$
    $loss = [Tar_i - Q(S, a, b)]^2$
    $Q* \leftarrow \min_Q (loss)$
**end while**
**return** $Q*$
```

In order to keep track of all relevant CVEs, we parsed a relevant data set containing a large amount of them. The data set contains valuable identifying information for CV, but what we want for a given CVE is specific courses of action for it and what each action mitigates. To parse this from a large file of 813 megabytes, we had to construct an algorithm that singles out courses of action and mitigation and stores them in pairs. In the file, the course of action would often function as a header with other identifying information about the CVE stored below it. One part of this identifying information is the other piece of information we are looking for, the mitigation section. A word-matching algorithm was employed to pull out the specific lines containing the course of action and mitigation portion, they were stored in a python dictionary with the course of action as the key and the mitigation as the value.

Once we parse the CKG to get the (*Course-of-Action*, *Attack-Pattern*) pairs and we find the CVEs associated with the *Course-of-Action*, we perform a simple word matching to map the *Course-of-Action* with the defender actions of our cyber battle environment.

Let us consider that the best-matched defender action is $b*$. Instead of randomly sampling from a uniform distribution of all the action sequences, we use a normal distribution as shown in Equation 4.

$$P_{exp}(b) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{b - b*}{\sigma}\right)^2\right) \quad (4)$$

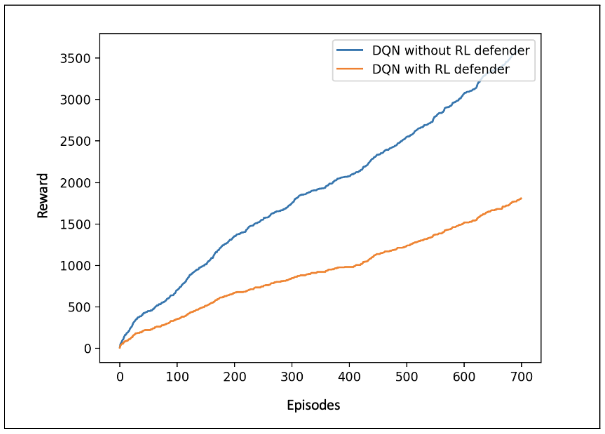
We first use an encoder to convert the defender actions to integer values. We set the mean of those integer values as $b*$. Initially, we set a low value for σ, so that we aggressively explore the favorable actions. As training progresses, we dilate the distribution by increasing the value of σ so that other actions are also explored. We also shuffle the 'unfavored' actions, to free them from the bias that can be imposed on them if they are numerically close to b*. This leads to a CKG-guided exploration, that helps in discovering valuable states early in the training process. We summarize our entire algorithm in III-C.

IV. Experimental Results

In this section, we discuss our experimental settings and findings. We perform two main types of experiments. We create a knowledge-guided minimax DQN to model a two-player game for cyber battle scenarios. Since CyberBattleSim already has an inbuilt DQN, we want to compare the performance of the vanilla DQN with our version. Secondly, we observe how knowledge guidance affects the performance of the respective algorithms.

In the first experiment, we created a baseline DQN model and a CKG-guided minimax DQN. The baseline DQN has a policy network and a target network. Each of these neural networks is a five-layered neural network with ReLU activations. We generate the simulation networks using CKGs as described in Section III-A. Our simulated network has 15 nodes, 7 of them having Windows operating systems of various versions and 8 of them having different versions of Linux. We run experiments for 12 episodes, and [700,800] iterations for each episode. We observe the performance of the DQN model with the CKG-guided minimax DQN model as represented in Figure 3. The cumulative attacker reward reduces greatly in the CKG-guided minimax DQN. The baseline DQN model does not have a defender, so the attacker is able to reach the attacker's goal unencumbered by the defender's actions. Hence, in the first case, the baseline DQN receives a much higher cumulative reward.

When we compare it with our algorithm, we see that the attacker's reward is reduced significantly. The attacker reward also gets reduced in later episodes. This can be attributed to the rules of the reward scores in CyberBattleSim. For example, an unsuccessful attempt at an exploit carries a very high negative reward (-50) for the attacker. In later episodes, with significantly higher exploitation, the defender takes appropriate reactive measures against an attacker's exploit resulting in the failure of the attack, which makes it even more difficult for the attacker to obtain positive rewards. The cumulative attacker rewards for all iteration ranges from 8.82 at iteration 0 to 1707.7 at iteration 700 in the DQN with the RL-based



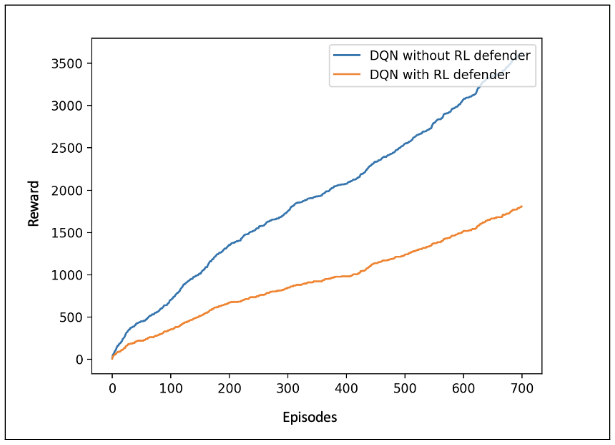


Fig. 3: Comparison of the cumulative attacker rewards for two scenarios. On the left we see the cumulative **attacker** rewards vs iterations for minimax DQN (DQN with RL defender) compared with a DQN with no defender. To the right we see the cumulative **attacker** rewards of a minimax DQN compared with a Random Agent

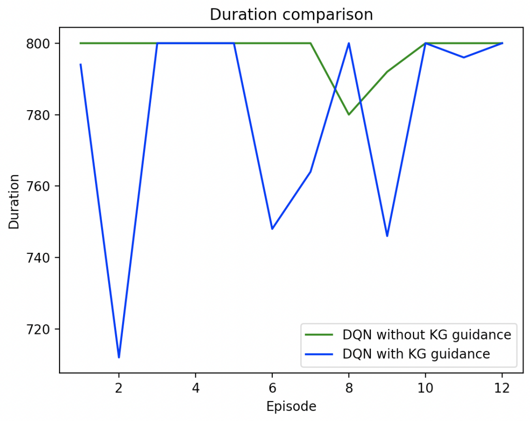


Fig. 4: Diagram showing the number of steps required to reach a goal at each episode with and without knowledge guidance

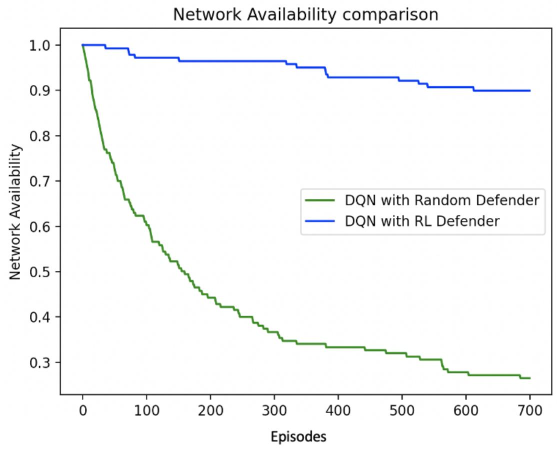


Fig. 5: Diagram showing the percentage of network available after defender agents perform actions

defender (minimax DQN), and it ranges from 12.2 to 3608.4 in the DQN with no defender.

However, our hypothesis was that with more effective defensive measures the attacker would also learn innovative ways to launch exploits. In order to check if the attacker agent has improved its performance, we compare the cumulative attack rewards of our CKG-guided minimax DQN with a Random agent. Figure 3 shows that our model has better attack rewards than the Random agent.

We also want to see how KG guidance has helped in reducing the time required to reach the end of an episode. An episode ends when either the attacker or the defender has reached their respective goals. The defender's goal is to evict the attacker from the network, and the attacker's goal is to own a certain percentage of the network. In Figure 4, we see that the DQN with CKG guidance reaches the goal faster at the beginning and at the end of the episodes. This can be attributed to our normal distribution based Exploration distribution $P_{exp}(b)$ that helps in identifying favorable actions.

It should be noted that after a certain number of iterations the normal distribution is relaxed, which makes other actions more likely to be explored. We see in one of the iterations (iteration 2) that the CKG guidance has helped the defender reach its goal faster (∼720 iterations). Towards the end, we see that in some iterations, DQN without CKG guidance has reached its goal faster. This is due to the CKG guidance only being used in the exploration phase. It does not carry a significant advantage during the exploitation phase, making it possible for DQNs without KG guidance to reach the goal faster.

CyberBattleSim also comes with a Defender that performs actions at random. We observe that the cumulative rewards for the attacker with the RL-based defender (minimax DQN) is similar to the cumulative attacker rewards of a DQN with a random defender. However, the random defender performs multiple actions in an iteration to bring down the attacker reward which drastically brings down the available network. We see the difference between the available network when a DQN with RL defender (minimax DQN) is performing actions and the available network when the DQN is against a random defender in Figure 5. This shows that the minimax DQN with CKG guidance takes accurate actions to bring down attacker rewards while keeping the network availability high. This is particularly important in the real-world scenario because automated defense mechanisms can render the network useless with the help of defensive actions. With the help of CKGs, we are able to keep most of the available network in place, while reducing the time required to train the RL algorithms.

V. Conclusion and Future Work

Our system based on *CyberBattleSim* uses a Cybersecurity Knowledge Graph (CKG) guided minimax Deep Q-Network (DQN) designed explicitly for cyber defense exercise (CDX) scenarios. The DQN reaches its goals faster and reduces the cumulative reward for the attacker because of our more robust defender. We also improve the system by including an RL-based defender. CKGs further improve the system because we include a method to incorporate their vulnerability information in the network graph properties. This also significantly reduces the human efforts required to describe a network for simulation.

The trained DQN can launch exploits in order to bypass the defender agent. The defender agent also gets simultaneously trained by the actions of the attacker agent and generates suitable actions for defense. With further improvement in attack simulations, it will be possible to uncover novel or zero-days through our model. In ongoing work, we are evaluating the performance of our models on real CDX datasets. Our knowledge graph-guided minimax DQN can predict how defense and attack teams perform in a CDX event which can then be compared to the actions of the human participants of the respective events. We are also extending this work of knowledge-guided exploration with more sophisticated methods to map CKG-suggested actions with RL actions.

Acknowledgements

This work was supported in part by funding from the National Security Agency and by National Science Foundation award number 2114892.

References

[1] Cybersecurity News. Why signature-based detection struggles to keep up with the new attack landscape? http://cybersecuritynews.com/signature-based-detection/, February 2022.

[2] Mandiant. Zero tolerance: More zero-days exploited in 2021 than ever before. http://mandiant.com/resources/zero-days-exploited-2021, April 2022.

[3] Forbes. Comparing legacy rules-based cybersecurity platforms and AI-based platforms. http://forbes.com/sites/forbestechcouncil/2022/02/14/comparing-legacy-rules-based-cybersecurity-platforms-and-ai-based-platforms, February 2022.

[4] Dan Goodin. Organizations are spending billions on malware defense that's easy to bypass. https://arstechnica.com/information-technology/2022/08/newfangled-edr-malware-detection-generates-billions-but-is-easy-to-bypass/, August 2022.

[5] Thanh Thi Nguyen and Vijay Janapa Reddi. Deep reinforcement learning for cyber security. *IEEE Transactions on Neural Networks and Learning Systems*, 2019.

[6] Microsoft. CyberBattleSim. https://github.com/microsoft/CyberBattleSim, April 2021.

[7] Sudip Mittal, Prajit Das, Varish Mulwad, Anupam Joshi, and Tim Finin. Cybertwitter: Using twitter to generate alerts for cybersecurity threats and vulnerabilities. In *Int. Conf. on Advances in Social Networks Analysis and Mining*. IEEE, 2016.

[8] Aditya Pingle, Aritran Piplai, Sudip Mittal, Anupam Joshi, James Holt, and Richard Zak. RelExt: Relation extraction using deep learning approaches for cybersecurity knowledge graph improvement. *IEEE/ACM Int. Conf. on Advances in Social Networks Analysis and Mining*, 2019.

[9] Aritran Piplai, Sudip Mittal, Anupam Joshi, Tim Finin, James Holt, and Richard Zak. Creating cybersecurity knowledge graphs from malware after action reports. *IEEE Access*, 2020.

[10] Sudip Mittal, Anupam Joshi, and Tim Finin. Thinking, fast and slow: Combining vector spaces and knowledge graphs. *arXiv preprint arXiv:1708.03310*, 2017.

[11] OASIS Open. Oasis cyber threat intelligence (CTI). http://oasis-open.github.io/cti-documentation, 2021.

[12] Sharmishtha Dutta, Nidhi Rastogi, Destin Yee, Chuqiao Gu, and Qicheng Ma. Malware knowledge graph generation. *arXiv preprint arXiv:2102.05583*, 2021.

[13] Nidhi Rastogi, Sharmishtha Dutta, Mohammed J Zaki, Alex Gittens, and Charu Aggarwal. Malont: An ontology for malware threat intelligence. In *International Workshop on Deployable Machine Learning for Security Defense*, pages 28–44. Springer, 2020.

[14] Aritran Piplai, Priyanka Ranade, Anantaa Kotal, Sudip Mittal, Sandeep Nair Narayanan, and Anupam Joshi. Using knowledge graphs and reinforcement learning for malware analysis. In *International Conference on Big Data*, pages 2626–2633. IEEE, 2020.

[15] Jing Liu, Yuan Wang, and Yongjun Wang. The similarity analysis of malicious software. In *Int. Conf. on Data Science in Cyberspace*. IEEE, 2016.

[16] Anand Patwardhan, Vlad Korolev, Lalana Kagal, and Anupam Joshi. Enforcing policies in pervasive environments. In *Int. Conf. on Mobile and Ubiquitous Systems: Networking and Services*, pages 299–308. IEEE, 2004.

[17] Nitin Kumar Sharma and Anupam Joshi. Representing attribute based access control policies in owl. In *2016 IEEE Tenth International Conference on Semantic Computing (ICSC)*, pages 333–336. IEEE, 2016.

[18] Tim Finin and Anupam Joshi. Agents, trust, and information access on the semantic web. *ACM SIGMOD Record*, 31(4):30–35, 2002.

[19] Sai Sree Laya Chukkapalli, Anupam Joshi, Tim Finin, Robert F Erbacher, et al. Capd: A context-aware, policy-driven framework for secure and resilient iobt operations. *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications IV, SPIE Defense+ Commercial Sensing*, 2022.

[20] Kelton AP da Costa, João P Papa, Celso O Lisboa, Roberto Munoz, and Victor Hugo C de Albuquerque. Internet of things: A survey on machine learning-based intrusion detection approaches. *Computer Networks*, 151:147–157, 2019.

[21] Daniele Ucci, Leonardo Aniello, and Roberto Baldoni. Survey of machine learning techniques for malware analysis. *Computers & Security*, 81:123–147, 2019.

[22] Derui Ding, Qing-Long Han, Yang Xiang, Xiaohua Ge, and Xian-Ming Zhang. A survey on security control and attack detection for industrial cyber-physical systems. *Neurocomputing*, 275:1674–1683, 2018.

[23] Aritran Piplai, Sai Sree Laya Chukkapalli, and Anupam Joshi. Nattack! adversarial attacks to bypass a GAN based classifier trained to detect network intrusion. In *Intl. Conf. on Big Data Security on Cloud, Intl. Conf. on High Performance and Smart Computing, Intl. Conf. on Intelligent Data and Security*, pages 49–54. IEEE, 2020.

[24] Wenyi Xu, Xiaofeng Gao, Yin Sheng, and Guihai Chen. Recommendation system with reasoning path based on DQN and knowledge graph. In *15th International Conference on Ubiquitous Information Management and Communication*. IEEE, 2021.

[25] Xinshi Chen, Shuang Li, Hui Li, Shaohua Jiang, Yuan Qi, and Le Song. Generative adversarial user model for reinforcement learning based recommendation system. In *Int. Conference on Machine Learning*, pages 1052–1061. PMLR, 2019.

[26] Patrick Phillips. Reinforcement learning in two player zero sum simultaneous action games. *arXiv preprint arXiv:2110.04835*, 2021.

[27] Liang Xiao, Xiaoyue Wan, Canhuang Dai, Xiaojiang Du, Xiang Chen, and Mohsen Guizani. Security in mobile edge caching with reinforcement learning. *IEEE Wireless Communications*, 25(3):116–122, 2018.

[28] Xin Xu and Yirong Luo. A kernel-based reinforcement learning approach to dynamic behavior modeling of intrusion detection. In *International Symposium on Neural Networks*, pages 455–464. Springer, 2007.

[29] Ahmad Ridley. Machine learning for autonomous cyber defense. *The Next Wave*, 22(1):7–14, 2018.

[30] Zhisheng Hu, Ping Chen, Minghui Zhu, and Peng Liu. Reinforcement learning for adaptive cyber defense against zero-day attacks. In *Adversarial and Uncertain Reasoning for Adaptive Cyber Defense*, pages 54–93. Springer, 2019.

[31] Jianqing Fan, Zhaoran Wang, Yuchen Xie, and Zhuoran Yang. A theoretical analysis of deep q-learning. In *Learning for Dynamics and Control*, pages 486–489. PMLR, 2020.

[32] Ariel Futoransky, Fernando Miranda, José Orlicki, and Carlos Sarraute. Simulating cyber-attacks for fun and profit. *arXiv preprint arXiv:1006.1919*, 2010.

[33] Jelena Mirkovic and Terry Benzel. Teaching cybersecurity with deterlab. *IEEE Security & Privacy*, 10(1):73–76, 2012.

[34] Kevin Schoonover, Eric Michalak, Sean Harris, Adam Gausmann, Hannah Reinbolt, Daniel R Tauritz, Chris Rawlings, and Aaron Scott Pope. Galaxy: a network emulation framework for cybersecurity. In *11th USENIX Workshop on Cyber Security Experimentation and Test (CSET 18)*, 2018.

[35] George Rush, Daniel R Tauritz, and Alexander D Kent. Coevolutionary agent-based network defense lightweight event system (candles). In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 859–866, 2015.

[36] Andres Molina-Markham, Ransom K Winder, and Ahmad Ridley. Network defense is not a game. *arXiv preprint arXiv:2104.10262*, 2021.

[37] Maxwell Standen, Martin Lucas, David Bowman, Toby J Richer, Junae Kim, and Damian Marriott. Cyborg: A gym for the development of autonomous cyber agents. *arXiv preprint arXiv:2108.09118*, 2021.

[38] Zareen Syed, Ankur Padia, Tim Finin, Lisa Mathews, and Anupam Joshi. UCO: A unified cybersecurity ontology. In *Artificial Intelligence for Cyber Security: Technical Report WS-16-03*. AAAI, 2016.

[39] MITRE. CVELIST project. http://cybersecuritynews.com/signature-based-detection/, May 2013.

[40] Robert A. Martin and Sean Barnum. Common weakness enumeration (CWE) status update. *Ada Letters*, XXVIII(1):88–91, April 2008.

[41] NIST. Common vulnerability scoring system. https://nvd.nist.gov/vuln-metrics/cvss.

[42] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, and Georg Ostrovski. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.