Using Knowledge Graphs and Reinforcement Learning for Malware Analysis

Aritran Piplai*, Priyanka Ranade*, Anantaa Kotal*, Sudip Mittal†, Sandeep Nair Narayanan‡, Anupam Joshi*
*Dept. of Computer Science & Electrical Engineering, University of Maryland, Baltimore County,
Email: {apiplai1, priyankaranade, anantak1, joshi}@umbc.edu
† Department of Computer Science, University of North Carolina, Wilmington, Email: mittals@uncw.edu
‡ Cisco Research and Development, sand7@umbc.edu

***Abstract*—Machine learning algorithms used to detect attacks are limited by the fact that they cannot incorporate the background knowledge that an analyst has. This limits their suitability in detecting new attacks. Reinforcement learning is different from traditional machine learning algorithms used in the cybersecurity domain. Compared to traditional ML algorithms, reinforcement learning does not need a mapping of the input-output space or a specific user-defined metric to compare data points. This is important for the cybersecurity domain, especially for malware detection and mitigation, as not all problems have a single, known, correct answer. Often, security researchers have to resort to guided trial and error to understand the presence of a malware and mitigate it.**

In this paper, we incorporate prior knowledge, represented as Cybersecurity Knowledge Graphs (CKGs), to guide the exploration of an RL algorithm to detect malware. CKGs capture semantic relationships between cyber-entities, including that mined from open source. Instead of trying out random guesses and observing the change in the environment, we aim to take the help of verified knowledge about cyber-attack to guide our reinforcement learning algorithm to effectively identify ways to detect the presence of malicious process names so that they can be deleted to mitigate a cyber-attack. We show that such a guided system outperforms a base RL system in detecting malware.

***Index Terms*—Reinforcement Learning, Knowledge Graphs, Cybersecurity, Artificial Intelligence**

I. Introduction

Cybersecurity aims to protect hardware and software components in a computer system against malicious attacks. A key part to protecting against malicious attacks is identifying them, as early as possible in the Cyber Kill Chain [7]. The most common approaches today are signature based, which of course can be defeated by adversaries by minor modifications of the malware or its dropper. Another approach is based on behavioural anomalies. This can be done by observing the system behavior over time and flagging any aberrant behavior. However, such anomaly detection based approaches often misidentify less frequently occurring legitimate system actions as attacks. Another commonly used approach is to use machine learning on network data to detect attacks [24]. In 2019 for instance, the IEEE Big Data conference organized a competition on the use of machine learning algorithms to detect attacks based on observed network data [8].

Machine learning is an important tool for recognising patterns in data. In the cybersecurity space, this can be useful to identify the behavior of a system under attack. However, standard machine learning algorithms have limitations in the cybersecurity space, especially in real deployments [28], [35]. In part, this is because the dataset is highly imbalanced – most of the observed data in a real system is not attacks, and unless a dataset is artificially curated to be balanced, it will mostly have benign data. In other words, the base rate of an attack is quite low. Hence, most machine learning algorithms tend to overfit for the data points of non-attack scenarios. Such learning algorithms are prone to weak performances in the real world. They also struggle in their ability to generalize, to unseen attacks.

There are also other problems in the cybersecurity space for traditional ML approaches. In other domains like natural language processing or computer vision, it is fairly simple for a human, who is not necessarily an expert in the specific task, to draw a conclusion the Artificial Intelligence (AI) system is trying to reach with high confidence. For example, a human may not need complicated background knowledge, except what is in implicit in the image itself to differentiate the image of a cat from a dog. However, looking at netflow data, it is not easy for a person to decide if it represents an attack – this requires significant expertise and background knowledge. So when we are trying to mimic the way security professionals reach a decision about a task, it is imperative that we consider a way to incorporate their prior knowledge that is not in the data itself into the machine learning system. This is because their prior knowledge and experience, as opposed to common knowledge, may dictate how they perform their tasks in their profession. For example, a security professional might take what they know about recent discussions in online forums about vulnerabilities in identifying an attack on their system. Also, in tasks like image recognition, there is not much scope for subjectivity for reaching a conclusion. Usually, an image is either a cat or not. However, what is an attack is not always clear, and for sophisticated APTs, even experts may not even recognize an attack as it is happening. It has been reported that often a third party identifies and APT attack, not the in house security team of the organization being attacked.

Reinforcement Learning (RL) mimics the way human beings tend to process new information. RL does not require sub-optimal actions to be explicitly corrected. Instead, it tries to find a balance between exploration (of new knowledge) and exploitation (of current knowledge). It does not assume

any prior mathematical modelling for the environment. This gives the RL algorithm more flexibility in learning about a new knowledge space.

In this paper, we make two key contributions. First, we use reinforcement learning for malware detection. Second, we also use knowledge mined from open information sources describing the same or similar malware attacks to change the behavior of the RL algorithm, automatically changing the parameters of the RL algorithm to adapt its reward functions and their initial probability distributions. This approach is not only a way to solve the issue of having to rely on pre-defined loss functions for traditional ML systems created by individuals who may not be experts in cybersecurity, it also helps to mimic how security professionals use their own knowledge in identifying attacks.

We organize our paper as follows - Section II talks about the key concepts of the different aspects of our algorithm. In Section III, we discuss our core algorithm. We discuss our findings in Section IV. We also discuss some of the relevant work conducted in this area in Section V, and we finally conclude our paper in Section VI.

II. Background

In this section, we discuss key reinforcement learning algorithm details and our general approach of representing extracted open source text in a Cybersecurity Knowledge Graph (CKG).

A. Reinforcement Learning

We utilize methods in model-free reinforcement learning in our approach [30]. Model-free RL agents employ prior experience and inductive reasoning to estimate, rather than generate, the value of a particular action. There are many kinds of model-free reinforcement learning models such as SARSA, Monte Carlo Control, Actor-Critic and Q-learning. We specifically utilize the Q-learning methods [30].

Q-learning agents learn an action-value function (policy), which returns the reward of taking an action given a state. Q-learning utilizes the Bellman equation in order to learn expected future rewards. We calculate the maximum future reward $max(Q(s', a'))$ given a set of multiple actions corresponding to different rewards. $Q(s, a)$ is the current policy of an action a from state s, and γ is the discount factor. The discount factor is the total reward an agent will receive from the current iteration until termination, and allows us to value short term reward over long term reward. The goal is to therefore maximize the discounted future reward at every iteration [32].

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

We update a policy table, also known as a Q-table for every action taken from a state. A Q-table is simply a lookup table that preserves the maximum expected reward for an action at each state. The columns represent actions and the rows represent states. The Q-table is improved at each iteration and is controlled by the learning rate α [32].

B. Cybersecurity Knowledge Graphs (CKGs)

Cybersecurity Knowledge Graphs have been widely used to represent Cyber Threat Intelligence (CTI). We use CKG to store CTI as semantic triples that helps in understanding how different cyber entities are related. This representation allows the users to query the system and reason over the information. Knowledge graphs for cybersecurity have been used before to represent various entities [23]. Open source CTI has been used to build CKGs and other agents to aid cybersecurity analysts working in an organization [16]–[18], [21], [27]. CKGs have also been used to compare different malware by Liu et al. [13]. Some behavioral aspects have also been incorporated in CKGs, where the authors used system call information [22]. Graph based methods have also been post processed by machine learning algorithms, as demonstrated by other approaches [2], [9], [10].

III. Methodology

In this section, we discuss the principles of our proposed algorithm. Figure 1, gives us an overview of the different aspects of our system. We provide text describing a piece of malware as an input to our cyber knowledge extraction pipeline and we receive a set of semantic triples as an output. We assert this set of triples to a CKG. We sometimes fuse this CKG with data from other sources such as behavior analysis [25]. This forms our knowledge base that will help us identifying malicious activity in a system and also suggest ways to mitigate them. The RL algorithm acts on the malware behavior data. We tune the parameters of the RL algorithm based on the information present in the CKGs that can be retrieved by querying the CKG.

A. Cyber Knowledge Extraction (CKGs) from prior knowledge sources

We have an established cyber knowledge extraction pipeline that takes malware After Action Reports and automatically populates CKGs [26]. The method uses a trained 'Malware Entity Extractor' to detect cyber- named entities. Once the malware entities have been detected, a deep neural network is used to identify the relationships between them [23]. The relationship extractor takes pairs of entities, generates their text embedding using a trained word2vec [15] model, and produces a relationship as an output for the pairs of entities. Finally the entity-relationship set is asserted into a CKG.

We use these trained models on open-source text describing the malware we use in our experiments, or similar malware. In order to find open source text analysing the malware, we use the known MD5 Hash of the malware and perform a web search to look for articles talking about the same malware.

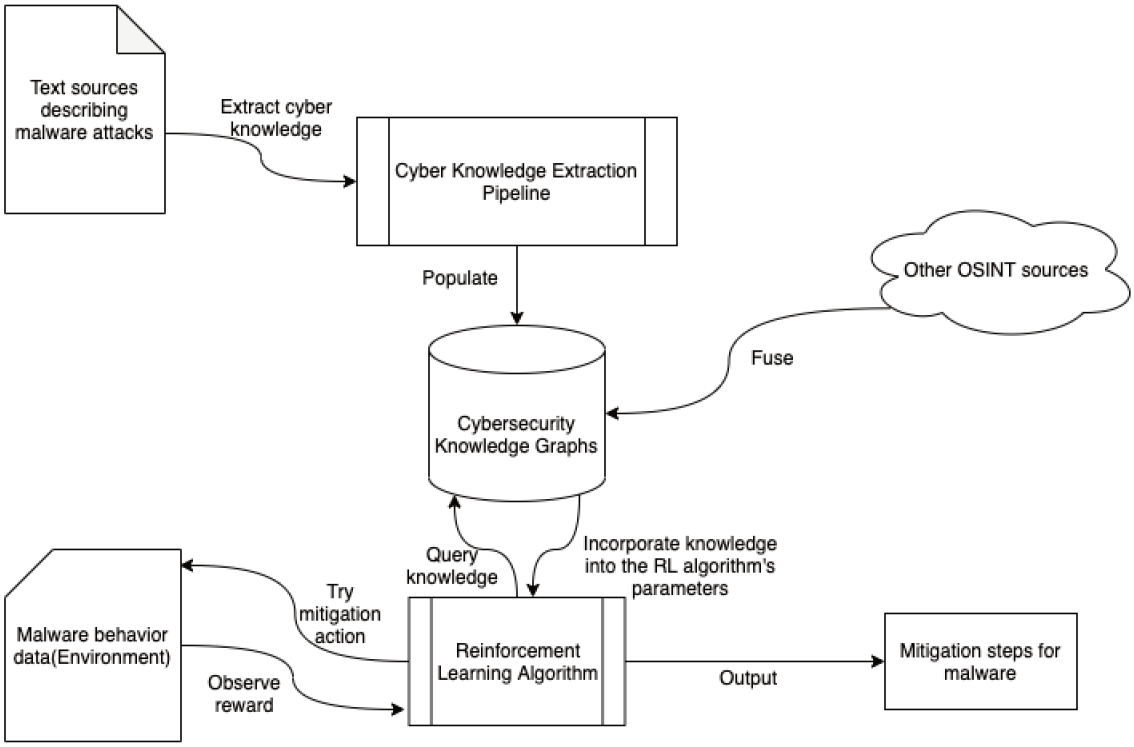


Fig. 1: An architecture diagram specifying the different steps of our proposed method

If we seek additional information about the malware samples, we can also search for open-source articles about the malware family if that is known. We produce the open-source text that we gather from the web search as an input to the cyber knowledge extraction pipeline. As a result, we get semantic triples describing the malware asserted in the CKG. Querying this CKG can result in entities that prove to be valuable for modelling our RL algorithm. This CKG populated with information about the malware acts as our prior knowledge source.

B. Reinforcement Learning Algorithm

We detonated malware samples in an isolated environment and observed some system parameters that represent the behavior of the malware sample. We used the same method to collect the malware behavior data as described by Piplai et al. [25]. The malware dataset comprises of samples downloaded from VirusTotal[1]. The data was collected over a time period of 60 minutes for each malware. We refer to the first 30 minutes the malware is not active, as the *benign phase*. The next 30 minutes form the *malicious phase*, where the malware is active in the system. Traffic was generated artificially to simulate multiple clients interacting with the malware infected system. Table I, describes the different parameters that were collected during the exercise. In our paper, we aim to use Reinforcement Learning to identify a sequence of malicious processes that may be created by the malware to perform the attack.

[1]VirusTotal. https://www.virustotal.com/

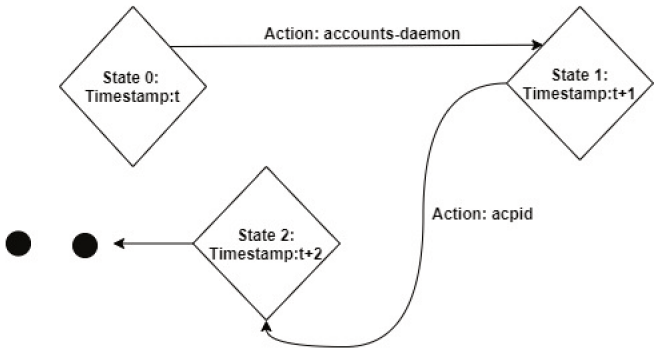
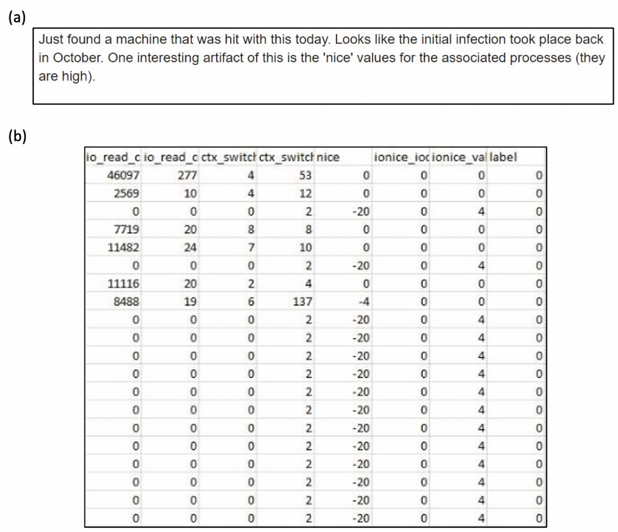


Fig. 2: Diagram showing the state transformation with respect to the actions in our dataset

We use Q-Learning for the purpose of identifying malicious process names. The behavior data that we collected resulted in data snapshots at a time interval of 10 seconds. For Q-Learning problems, we define a state space and an action space. To define a state-space we ideally need to look at key identifiers that can be used to isolate a row in our data. We use the 'timestamp' for this purpose, and that helps in creating as many states as there are rows in our data. For a single experiment, we have 26000 to 28000 rows. We define our action space by the total number of distinct process names that are getting created in a single experiment. Since this number may vary per experiment, we take a superset of all the processes that are created for all our experiments. In our experiments, we see that a total of 99 distinct processes are created over the course of the time of data collection. We use

TABLE I: Virtual machines performance metrics [1].

Metric Category	Description
Status	Process status
CPU information	CPU usage percent, CPU times in user space, CPU times in system/kernel space, CPU times of children processes in user space, CPU times of children processes in system space.
Context switches	Number of context switches voluntary, Number of context switches involuntary
IO counters	Number of read requests, Number of write requests, Number of read bytes, Number of written bytes, Number of read chars, Number of written chars
Memory information	Amount of memory swapped out to disk, Proportional set size (PSS), Resident set size (RSS), Unique set size (USS), Virtual memory size (VMS), Number of dirty pages, Amount of physical memory, text resident set (TRS), Memory used by shared libraries, memory that with other processes
Threads	Number of used threads
File descriptors	Number of opened file descriptors
Network information	Number of received bytes, Number of sent bytes



io_read_c	io_read_c	ctx_switcl	ctx_switcl	nice	ionice_ioc	ionice_val	label
46097	277	4	53	0	0	0	0
2569	10	4	12	0	0	0	0
0	0	0	2	-20	0	4	0
7719	20	8	8	0	0	0	0
11482	24	7	10	0	0	0	0
0	0	0	2	-20	0	4	0
11116	20	2	4	0	0	0	0
8488	19	6	137	-4	0	0	0
0	0	0	2	-20	0	4	0
0	0	0	2	-20	0	4	0
0	0	0	2	-20	0	4	0
0	0	0	2	-20	0	4	0
0	0	0	2	-20	0	4	0
0	0	0	2	-20	0	4	0
0	0	0	2	-20	0	4	0
0	0	0	2	-20	0	4	0
0	0	0	2	-20	0	4	0
0	0	0	2	-20	0	4	0
0	0	0	2	-20	0	4	0
0	0	0	2	-20	0	4	0

Fig. 3: Diagram showing a knowledge source (a) describing a similar malware and the time series data we collected after detonating a malware (b).

99 as the action space for our experiments.

Intuitively, we can think of a single process created at timestamp 't' is changing the state of the system. The new state is identified with the timestamp 't+1'. The system features change as the new process is created, and we aim to identify the set of processes responsible for state changes that might signify a malicious attack. Figure 2, demonstrates an example how we model process names as action spaces and identify different states with timestamps. We use this intuition to create reward functions that may help us identify malicious processes. For example, a high deviation from restful network activity after the creation of a process may signify the presence of a malware in the system. A surge in Input/Output (I/O) read/write bytes may also mean the same. We use this intuition to form reward functions.

C. Utilizing CKGs in the RL algorithm

As stated in Section I, in a system that calls for complicated analysis to reach a conclusion, we need to consider expert knowledge that may help us better identify or mitigate malicious process names. In Figure 3, we can see a screenshot of an open source text description of a malware sample that an expert has provided. Instead of relying on hand-crafted reward functions to evaluate the quality of a state, we can use the knowledge extracted from open source text and use them directly in the reward function. In Figure 3, the text description serves as input to a CKG that captures the semantic relationship between the malware and the 'high nice values'. The 'high nice values' are an indicator for the malware referenced. The CKG establishes a relationship between the *Malware* and the *Indicator*. When we query the knowledge graph about the indicator, it returns 'high nice values' as a result. We can map this entity to features of our behavior data and incorporate them into the parameters of our RL algorithm.

There are two ways we incorporate prior knowledge into our RL algorithm. The first method are inspired on some of the concepts mentioned by Moreno et al. [19] In Q-Learning, we have an exploration phase and an exploitation phase. In vanilla Q-Learning, exploration phases, we try random actions and observe the reward. During the exploitation phase, we choose the action that maximizes the Q-value for the state-action pair. Moreno et al. [19] demonstrate a method of using existing Q-values of a given state action pair for exploration leading to a faster convergence. We can see this in Equation 1. We can also manipulate the exploration probabilities with the help of the *T(s)* value.

$$P(s, a_i) = \frac{\exp(Q(s, a_i)/T(s))}{\sum_j \exp(Q(s, a_j)/T(s))} \tag{1}$$

In our algorithm, we use the extracted knowledge to tune the probability distribution for exploration of a state-action pair. For example, Equation 2 will change the probability distribution, into assigning higher probabilities for actions associated with a 'nice value' of -20. The extracted knowledge from expert sources show us 'nice values' are important while searching for malicious processes.

$$P(s, a_i) = 1 - \frac{(nicevalue(a_i) + 20)}{2 \cdot \mid \max_j nicevalue(a_j) \mid} \tag{2}$$

The second method is incorporating the extracted knowledge into our reward functions to identify the malicious processes. For example, if a knowledge source indicates that

the processes create multiple threads, we can use that as an additional parameter for our reward function. We will discuss the different reward functions that we have constructed from the knowledge sources and their performance in Section IV.

IV. Experimental Results

In this section, we discuss the preliminary results from our experiments. Most of the malware analysis and machine learning research concentrates on evaluating the performance of the machine learning algorithm on a dataset of malware samples. For example, Gavrilut et al. [6], discuss multiple machine learning algorithms to detect malware files using perceptrons and kernelized perceptrons and they evaluate the performance of their trained algorithm on a test set. Since our approach is aimed at discovering a sequence of process names that we suspect to be malicious, we aim to rank the actions (processes) with respect to their q-values after some episodes of training. To be precise, we use 140 episodes for training with 10,000 steps in each of them. The high number of step count compared to the number of episodes is because the time series data consists of 26,000 to 28,000 states. If we use a small value for the step count, we would be able to cover only a small portion of the state space in one episode. Hence, we keep the step count high and the episode count relatively low. We aim to calculate as much q-values as possible in one episode itself. We then record how our RL system scores the known malicious processes with respect to other processes that are benign.

We use a combination of reward functions from Equations 3, 4, and 5. The first reward function is constructed based on common knowledge and intuition. If the I/O activity decreases or the network activity after a process creation, we can make an educated assumption that the process could be benign. We assign a reward value of '+1' if the I/O or network activity decreases after a process creation. For this we calculate the average of I/O read/write bytes for 5 time-steps before a process creation and 5 time steps after a process creation. The same approach is used for KiloBytes (KB) Sent/Received. This is done because the effect of a process creation may not be immediate. In our first experiment (Exp 1) we simply use Equation 3 and Equation 4 as our reward function.

In our second experiment (Exp 2), we keep the reward functions constant. However, we change the exploration criteria for our RL algorithm based on prior knowledge. The exploration probability distribution is stated in Equation 2. This helps the RL algorithm to explore state-action pairs that have high priority nice values leading to a faster convergence.

$$Reward(s,a) \quad += 1 \quad \text{(if I/O write/read decreases or KB sent/recieved decreases)} \tag{3}$$

$$Reward(s,a) \quad -= 1 \quad \text{(otherwise)} \tag{4}$$

$$\begin{aligned} Reward(s,a) = w1 \cdot \frac{nicevalue}{20} \\ + w2 \cdot (numthreads(state, action) \\ - numthreads(state-1, previousaction)) \end{aligned} \tag{5}$$

In the third experiment (Exp 3), we incorporate the prior knowledge source in our reward function. The prior knowledge source states that the nice values of the associated processes are high priority. We use Equation 5 with the value of 'w2' set to 0. A high priority nice value will be close to -20. So, a state-action pair for which the nice value is close to -20, will have a negative reward associated with it if the value of 'w1' is set to 1.

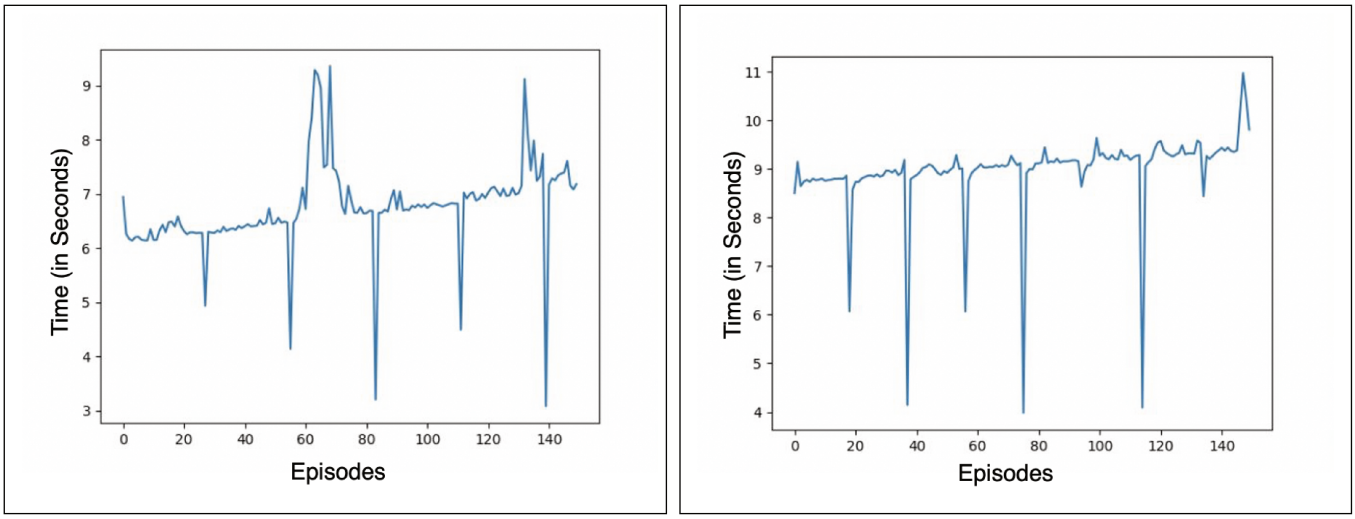
In the last experiment (Exp 4), we select another source of information describing the Bill Gates Botnet family. The source tells us that this malware family spawns a significantly higher number of threads. We use Equations 3, 4, and 5 for this experiment. We use a weighted sum of the two prior sources in this experiment.

Type	Exp-1	Exp-2	Exp-3	Exp-4
Q-value	-5.1	-5.7	-7	-16.99
Rank	9(99)	11(99)	8(99)	1(99)

TABLE II: Ranking and Q-values of the known malicious process

In Table II we can see that the reward function using the weighted mean of prior information sources is able to identify the known malicious process as the highest ranked process. The Q-values are greater in magnitude because the reward functions have more parameters included in them. This shows that including more knowledge sources in our RL algorithm yields better results.

In Figure 4, we can see the comparison of time required to complete each episode that consists of 10,000 steps. We argued previously that tuning the exploration probability distribution would lead to a faster convergence. We observe that this is partly true. In sharp dips signify the exploitation phase of the RL algorithm, when the algorithm knows what it looks for. The surge signifies more time to find 'actions' during exploration that fit the state-transition. Although the average time is lower in Figure 4a than in Figure 4b, this is because in the beginning the environment is benign. So, the tuned probability distribution that is supposed to aid in the process of finding malicious processes hinders the RL algorithm to find processes or actions, that are benign in the beginning, that fits the state transformation. However, we do observe some sharp peaks in Figure 4a, as the RL algorithm's episodes move to the malicious phase. In contrast, the time per episode is more consistent in Figure 4b. This is the result of the changed probability distribution that helps the RL algorithm in finding processes faster in the malicious phase.



(a) For Exp 1 (user-defined reward functions)

(b) For Exp 2 (tuned exploration probabilities)

Fig. 4: Comparison of time required to complete each episode for two experiments

V. Related Work

Security is critical in maintaining the integrity of cyber systems. Technological innovations have lead to complex system architectures that are increasingly hard to protect. Preemptively identifying attack scenarios and taking mitigating actions is a complex task. There is no perfect solution to it yet. AI, especially ML, can be helpful in answering questions to which there is no easy answer. ML techniques have been previously used in the domain of cybersecurity for intrusion detection [3], malware detection [31], and cyberphysical attacks [4]. However, traditional ML techniques are not ideal for to emulate how a security researcher conducts malware analysis. The shortcomings of traditional ML techniques in the domain of cybersecurity have been discussed in Section I.

RL is an alternate strategy for automatic resolution of tasks in domains where the correct answer is not known. This technique is popular in the field of game playing [12], robotics [11] and even for biological data [14]. There are similarities in the problem space of game playing and attack identification and mitigation in the cybersecurity domain. Previous literature explore how applications of RL can be applied to various aspects of cybersecurity. The 2016 study by Feng et al. [5] characterises cyber state dynamics as a function of physical state, control inputs, disturbances, and current cyber attacks and defenses. The cyber defense problem was then modeled as a two-player zero-sum game. An RL algorithm was used to efficiently learn the optimal cyber defense strategy in the problem space. Some of these models generate adversaries to train the RL algorithm by changing some of the parameters of the malware sample that may make it lose its potency. In our experiments, we use the behavior of actual malware samples. We do not automatically generate adversaries yet as there are challenges to preserve the malware potency.

Robustness in the cyber-physical space measures the resiliency of a given specification being satisfied. The security problem is to find candidates with minimal robustness (counterexample) by falsification or changing of input and parameters of the system. Conventional methods, such as simulated annealing and cross entropy, require a large number of simulation runs to the minimize robustness. Akazaki et al. [34] proposed the use of RL techniques, i.e., Asynchronous Advantage Actor-Critic (A3C) and Double Deep Q Network (DDQN), to reduce the number of simulation runs required to find such counterexamples. Intrusion detection is also an important defense technique and current techniques leave room for improvement. Xu et al. [33] proposed a kernel-based RL approach using Least-Squares Temporal-Difference (LS-TD) for intrusion detection that showed better accuracy than Hidden Markov Model and and linear TD algorithms. Shamshirband et al. [29] discuss the challenges in detecting malicious behavior in Wireless Sensor Networks (WSNs). They explain why traditional intrusion detection methods fail in to detect distributed denial-of-service attacks and propose a Game-based Fuzzy Q-learning (G-FQL) algorithm that combines game theoretic approach and the fuzzy Q-learning for intrusion detection in WSN.

Although these papers use RL for malware analysis, these approaches do not take advantage of prior knowledge about the domain when developing an attack detection and mitigation strategy. The inability to use pre-existing knowledge, puts prior RL algorithms for cyber-security at a significant disadvantage. The AI has very limited idea in the beginning about the environment the malware is acting on, and the possible actions that a human, who is an expert in this domain, would take. The same disadvantage burdened the RL algorithms for mobile robotics. In 2004, Moreno et al. [20] proposed a supervised reinforcement learning for mobile robotics that takes advantage of external knowledge and validates it in a “wall-following” behaviour. Although the paper broadly discussed about ranking the ‘prior human knowledge sources’ by changing the explo-

ration probability distribution, it also briefly discussed about how it can be used to improve performance. In our study, we use a variant of this technique in one of our experiments. The key difference is that we are utilizing CKGs that already hold knowledge about cybersecurity and we do not have to rely on expensive human inputs.

VI. Conclusion and Future Work

In this paper, we propose using RLs to detect malware, and incorporating prior knowledge from CKGs into the RL based detection system. This separates our approach from traditional use of ML approaches to detect attacks. Our approach mimics the way cybersecurity professionals in a SoC analyze the reported sensor data based on their backgroud knowledge to see if the system is currently under attack by a malware. Sometimes they resort to guided trial and error method (manifested in this case by the RL algorithm), and in some cases they use their own knowledge and experience to derive conclusions.

Specifically, in our experiments, we show how prior knowledge taken from text sources describing a malware activity can be used in an RL algorithm to detect malicious process. We suggest that deleting these processes may prove to be an acceptable mitigation strategy. However, the knowledge stored in the CKGs can actually provide multiple mitigation strategies that can be used as a malware executes. In ongoing work, we are using multiple known mitigation strategies after detonating a malware sample. This can produce better mitigation strategies for a malicious sample. We are also using the malware features to identify the malware family that a new malware sample belongs to. We can the use open source intelligence about that malware family to formulate candidate mitigation steps. This will help us build a robust mitigation strategy generator that will be able to use and integrate the knowledge of security researchers with RL, to yield the best sequence of steps needed for a particular malware attack to be defeated.

Acknowledgement

The authors would like to thank Dr. Mahmoud Abdelsalam and Dr. Maanak Gupta for the dataset used in this work.

This work was supported by a United States Department of Defense grant, a gift from IBM research, and a National Science Foundation (NSF) grant, award number 2025685.

References

[1] Mahmoud Abdelsalam, Ram Krishnan, Yufei Huang, and Ravi Sandhu. Malware detection in cloud infrastructures using convolutional neural networks. In *11th Int. Conf. on Cloud Computing*. IEEE, 2018.

[2] Blake Anderson, Daniel Quist, Joshua Neil, Curtis Storlie, and Terran Lane. Graph-based malware detection using dynamic analysis. *Journal in Computer Virology*, 7(1):247–258, 2011.

[3] 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.

[4] 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.

[5] Ming Feng and Hao Xu. Deep reinforecement learning based optimal defense for cyber-physical system in presence of unknown cyber-attack. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8. IEEE, 2017.

[6] D. Gavriluţ, M. Cimpoeşu, D. Anton, and L. Ciortuz. Malware detection using machine learning. In *2009 International Multiconference on Computer Science and Information Technology*, pages 735–741, 2009.

[7] Eric Hutchins, Michael Cloppert, and Rohan Amin. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. https://www.lockheedmartin.com/content/dam/lockheed-martin/rms/documents/cyber/LM-White-Paper-Intel-Driven-Defense.pdf.

[8] A. Janusz, D. Kałuza, A. Chadzyńska-Krasowska, B. Konarski, J. Holland, and D. Ślezak. Ieee bigdata 2019 cup: Suspicious network event recognition. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 5881–5887, 2019.

[9] Karuna P Joshi, Aditi Gupta, Sudip Mittal, Claudia Pearce, and Tim Finin. Alda: Cognitive assistant for legal document analytics. In *AAAI Fall Symposium on Cognitive Assistance in Government and Public Sector Applications*. AAAI Press, 2016.

[10] Maithilee Joshi, Sudip Mittal, Karuna P Joshi, and Tim Finin. Semantically rich, oblivious access control using abac for secure cloud storage. In *Int. conf. on edge computing*, pages 142–149. IEEE, 2017.

[11] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[12] Robert Levinson. General game-playing and reinforcement learning. *Computational Intelligence*, 12(1):155–176, 1996.

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

[14] Mufti Mahmud, Mohammed Shamim Kaiser, Amir Hussain, and Stefano Vassanelli. Applications of deep learning and reinforcement learning to biological data. *IEEE transactions on neural networks and learning systems*, 29(6):2063–2079, 2018.

[15] Tomas Mikolov, Ilya Sutskevarand Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *26th International Conference on Neural Information Processing Systems - Vol. 2*, pages 3111–3119. ACM, 2013.

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

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

[18] Sudip Mittal, Anupam Joshi, and Tim Finin. Cyber-all-intel: An ai for security related threat intelligence. *preprint arXiv:1905.02895*, 2019.

[19] D. Moreno, Carlos V. Regueiro, R. Iglesias, and S. Barro. Using prior knowledge to improve reinforcement learning in mobile robotics. 2004.

[20] David L Moreno, Carlos V Regueiro, Roberto Iglesias, and Senén Barro. Using prior knowledge to improve reinforcement learning in mobile robotics. *Proc. Towards Autonomous Robotics Systems. Univ. of Essex, UK*, 2004.

[21] Lorenzo Neil, Sudip Mittal, and Anupam Joshi. Mining threat intelligence about open-source projects and libraries from code repository issues and bug reports. In *Int. Conf. on Intelligence and Security Informatics*. IEEE, 2018.

[22] Younghee Park, Douglas Reeves, Vikram Mulukutla, and Balaji Sundaravel. Fast malware classification by automated behavioral graph matching. In *6th Annual Workshop on Cyber Security and Information Intelligence Research*. ACM, 2010.

[23] 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. In *Int. Conf. on Advances in Social Networks Analysis and Mining*. IEEE, 2019.

[24] A. Piplai, S. S. L. Chukkapalli, and A. Joshi. Nattack! adversarial attacks to bypass a gan based classifier trained to detect network intrusion. In *2020 IEEE 6th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS)*, pages 49–54, 2020.

[25] Aritran Piplai, Sudip Mittal, Mahmoud Abdelsalam, Maanak Gupta, Anupam Joshi, and Tim Finin. Knowledge enrichment by fusing representations for malware threat intelligence and behavior. *IEEE International Conference on Intelligence and Security Informatics (ISI)*, 2020.

[26] 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*, 2020.

[27] Priyanka Ranade, Sudip Mittal, Anupam Joshi, and Karuna Joshi. Using deep neural networks to translate multi-lingual threat intelligence. In *Int. Conf. on Intelligence and Security Informatics*. IEEE, 2018.

[28] Maheshkumar Sabhnani and Gursel Serpen. Why machine learning algorithms fail in misuse detection on kdd intrusion detection data set. *Intelligent Data Analysis*, 2004.

[29] Shahaboddin Shamshirband, Ahmed Patel, Nor Badrul Anuar, Miss Laiha Mat Kiah, and Ajith Abraham. Cooperative game theoretic approach using fuzzy q-learning for detecting and preventing intrusions in wireless sensor networks. *Engineering Applications of Artificial Intelligence*, 32:228–241, 2014.

[30] Sutton and Barto. Reinforcement learning: An introduction. MIT press, 2018.

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

[32] Watkins and Dayan. Q-learning. In *Machine Learning)*, pages 279–292, 1992.

[33] 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.

[34] Yoriyuki Yamagata, Shuang Liu, Takumi Akazaki, Yihai Duan, and Jianye Hao. Falsification of cyber-physical systems using deep reinforcement learning. *IEEE Transactions on Software Engineering*, 2020.

[35] Roman V. Yampolskiy. Artificial intelligence safety and cybersecurity: a timeline of ai failures. *arxiv*, 2016.