ABSTRACT

Title of dissertation: KNOWLEDGE GRAPHS AND REINFORCEMENT LEARNING: A HYBRID APPROACH FOR CYBERSECURITY PROBLEMS
Aritran Piplai, Doctor of Philosophy, 2023

Dissertation directed by: Professor Anupam Joshi
Department of Computer Science and Electrical Engineering

With the explosion of available data and computational power, machine learning and deep learning techniques are being increasingly used to solve problems. The domain of cybersecurity is no exception, as we have seen multiple papers in the recent past using data-driven machine learning approaches for different tasks.

Rule-based and supervised machine learning-based approaches are often brittle in detecting attacks, can be defeated by adversaries that adapt, and cannot use the knowledge of experts. To address this problem, we propose a novel approach for cybersecurity tasks that leverages the supply of 'explicit' knowledge expressed as Knowledge Graphs, as well as the data-driven approaches of machine learning to ascertain the 'tacit' knowledge. The inspiration for this hybrid model is drawn from the manner in which security analysts work, combining their background knowledge with observed data from host and network-based sensors.

First, we work on extracting background knowledge from unstructured textual descriptors of malware. Next, we focus on the tasks of malware detection and mitigation policy generation. We are specifically interested in the capability of synthesizing novel ways in which a malware may carry out an attack that can be further used to detect unknown attacks. We combine the background or explicit knowledge with explorations in the "action space" of malware detection and malware mitigation agents.

Analysts take response strategies for novel malware based on their knowledge of past experiences, while also exploring new strategies through "trial and error". In this dissertation, we describe knowledge graph construction techniques from open-source text as well as several Reinforcement Learning (RL) based algorithms, guided by explicit background knowledge encoded in a knowledge graph, that best mimics the approach of security analysts. We observe that the efficiency of RL algorithms has increased by 4% by incorporating prior knowledge. We also observe that in simulated environments RL policies are able to generate more precise mitigation strategies with the help of prior knowledge. Certain parameters that measure the precision of mitigation actions, such as network availability, show an increase of 50% when prior knowledge is used.

KNOWLEDGE GRAPHS AND REINFORCEMENT LEARNING: A HYBRID APPROACH FOR CYBERSECURITY PROBLEMS

by

Aritran Piplai

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, Baltimore County in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2023

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction and Motivation

1.1 Introduction

Cybercrime is a growing menace and it has shown a growth of over 600% during the pandemic [1] years. The financial cost of cybercrime has been rising consistently by over 15% every year, and some estimates suggest that it will reach \$ 10 trillion by 2025 [2]. The Global Risk report [2] stated that even in the US in 2020, the detection rate is 0.05 percent. However, the silver lining is that with each passing day a greater number of dedicated security research groups are developing innovative ways to tackle cybercrime. The increased interest in solving cybersecurity problems is for the following reasons.

With vulnerabilities present in the software that people use, more sensitive information is exposed for attackers to collect. Some software vulnerabilities can also help attackers to gain control of the system. As our society becomes increasingly reliant on digital platforms and services, maintaining trust in online environments is paramount. By detecting these attacks, private companies deter potential offenders and ensure the safety of the users. Cybercrimes can also have severe implications for national security [3]. Detecting and preventing cyberattacks on critical infrastructure, government networks, or military systems is crucial to protect a nation's interests. It helps preserve the integrity and functionality of vital systems, ensuring the safety and stability of the country.

Relying on signature-based methods to detect cyber-attacks can prove to be unhelpful as 99% of malware programs are used only once in their current form before being modified [4]. Such modifications can render signature-based defenses useless. With more corporations adopting the use of machine learning (ML), we see a similar trend in cybersecurity as well [5]. ML 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, supervised ML models have limitations in the cybersecurity space, especially in real deployments [6, 7]. They also suffer from many of the same shortcomings as signature-based systems, since they are typically trained on historical data and have limited ability to generalize or foresee mutations in the training data. 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 [8]. 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 RL, we simulate these attack scenarios to create defenses that are more generic and do not rely on a specific trigger. RL models are constrained by partial observations of cyber threats, limiting their effectiveness. Malware behavior varies across environments, making it impossible to simulate all possibilities. External knowledge from domain experts plays a vital role by providing key information to RL algorithms during the learning process. This external knowledge is derived from open-source text describing cyber-attacks that we capture and represent in the form of a knowledge graph.

There is a plethora of open-source text available that talks about cyber-attacks and defense strategies. This is because the exchange of information brings numerous benefits to the field of cybersecurity [9]. Collaborating with others allows organizations, both private enterprises and government agencies, to develop a robust cybersecurity strategy, which serves as the most effective defense against malicious attacks. Open-source knowledge about the recent developments in cybersecurity have helped both defenders and adversaries. MIT published a report [10] about how a security company BitDefender announced that they found loopholes in a ransomware built by a group called the 'DarkSide'.

Mining information from these sources can help us extract key attack indicators and defensive measures. These help both security analysts as well as RL models as they simulate variants of attacks and determine the best course of action to mitigate these attacks simultaneously.

Open-source text about past cyber-incidents, as well as general information or common knowledge can be helpful for detecting cyberattacks. A centralized knowledge base that records static analyses of malwares, like VirusTotal [11] can also be helpful. Another contribution of our work, which we will discuss more about in Sections 5.1 and 5.2, is collecting information from unstructured sources with the help of Natural Language Processing (NLP). Malware analysts, in order to generate more information about cyber-attacks, collect malware samples, detonate them in a ‘sandbox’ and publish their findings in the form of unstructured text. We have fused both behavioral information as well as information extracted with NLP tools to be used by RL algorithms.

For specific tasks, security analysts have the option of querying from these CKGs to ascertain particular actions that they need to take to tackle the problems associated with these tasks. A few examples of these tasks can be malware identification, malware classification and malware mitigation. Since CKGs capture semantic relationship between different ‘entities’ of interest that are helpful for the above mentioned tasks, we propose to use them in our ML-based algorithms. Particularly, we choose reinforcement learning for these tasks. Reinforcement learning has the ability to emulate the way humans reach a conclusion, by trial and error. In certain cybersecurity tasks, it is difficult to fit a mathematical equation to minimize a pre-defined loss function. This is how most of the data-driven approaches work in cybersecurity. For example, a loss function would include network activity as a parameter if the malware uses command and control infrastructure as its attack pattern. However, with novel attacks being more in vogue, it becomes difficult to formulate loss functions to detect specific types of malware. For continuously evolving adversaries, it is helpful if we incorporate the CKG information about similar attack-patterns. A more natural approach would be to assign what states are known ‘good’ or ‘benign’ states and what states are indicative of the presence of a malware. This can be achieved through RL, as we have the option of assigning rewards for specific state,action pairs. RL algorithms can work particularly well for cybersecurity [12] because they can leverage their exploration capabilities to discover previously unknown attack and defense scenarios. This is an improvement over signature-based and supervised ML models that do not have the capability of envisioning intelligent mutations of previously known attacks and unknown attacks.

In this dissertation, we use the knowledge about cyber-entities asserted in CKGs for helping the RL algorithms designed for specific cybersecurity tasks. The RL algorithms receive inputs from the CKGs that guide them in evaluating state-action sequences, as well as state exploration. The cybersecurity tasks that we have chosen to address are detection, mitigation, and adversarial malware generation. For all these tasks, CKGs contain specific information that will be helpful. We evaluate the performance of these models in simulated environments, as well as in generation policies for detection for real malware data.

The thesis underlying our work is that *Reinforcement Learning combined with Knowledge Graphs will improve malware detection, as well as automatic threat-mitigation strategy generation.*

1.2 Motivation

Conventional ML models have found success in the domain of cybersecurity, both in theory and in practice. A few years back, Microsoft publicly stated how their ML based component of their 365 Defender was successful in identifying malicious files during the Emotet outbreak [13]. They used a simple light-weight ML model that looks at features such as 'entropy', 'number of sections' of a file and sends a classification signal to a more complex ML model running in the cloud. These ML models are successful when they are able to discover ‘tacit’ knowledge by simply looking at the data. They are also useful when the signature-based rules that could have also been used for the same purpose become very complex. However, if the objective of the ML model is to alter a few files to mitigate a perceived threat, it would not be obvious to the ML model that there are certain files in the system that should not be tampered with. For example, it is common knowledge for cybersecurity professionals that the Master Boot Record holds key information about the disk, and should not be tampered with. This particular knowledge is ‘explicit’ knowledge, and we do not need further exploration to verify this.

Recently in the AI community, there has been a renewed interest in a discussion on the differences between ‘tacit’ and ‘explicit’ knowledge [14]. There are certain pieces of information that an AI agent can discover on its own by looking at the data, and then there are other pieces of information that the AI agent may not be able to learn from the data. The data that the AI agent can learn from the data itself, is defined as the ‘tacit’ knowledge. However, some information cannot be inferred as easily from the data and there is a possibility that this information needs to be externally provided to the AI agent. This is what is defined as ‘explicit’ knowledge. Tacit knowledge is something that cannot be verbally communicated over, but explicit knowledge can be.

Kambhampati, in this opinion piece [14], argues that unfair importance has been put on data-driven learning for AI agents. This is paradoxical as humans do not learn that way. AI systems have been built for complex tasks, without any understanding of the intrinsic knowledge that is required to achieve those tasks.

Data-driven machine learning can uncover hidden facts, but it is also possible that it is a more cumbersome approach to realize already known facts. An example that has been used, is creating an AI agent to solve the Rubik's cube problem from billions of examples, rather than using the already established rules of Rubik's cube. A similar cyber-security specific argument can be made that we do not need a million sample points to find out how security analysts check for persistence. Looking at registry keys should be the first step in this regard.

Although there are two schools of thought that exists in this domain and one school of thought evidently talks about how everything can be discovered from data [15], it makes sense to utilize the already existing rules for cybersecurity for cybersecurity tasks. If we compare a task like image recognition (detection of cats) with malware detection, a deep neural network can discover features that indicate that the image is that of a cat. For malware detection, however, a deep neural network may be able to achieve a high accuracy at detection, but ignoring the rules set for those specific malware may result in using features that are irrelevant for the malware. Malware detection for humans is a far more challenging task, compared to identifying cats. Although data-driven approaches have the option to uncover unknown ways to detect a malware, it is difficult to say if a good performance on a test dataset indicates that the AI algorithm reasonably came to the same conclusion a human did. We want to use the rules set for malware detection, from cybersecurity knowledge graphs that contain information about past attacks that have been similar in nature.

However, we do not want to restrict our AI algorithm to only use the rules or identifiers that have been generated for previous attacks. We also want to leverage the capabilities of data-driven approaches to uncover unknown rules that may be beneficial for cybersecurity tasks, such as malware detection. We propose a hybrid model, that uses the rules for cybersecurity tasks, that we can garner from CKGs, and utilizes them in the data-driven approaches for cybersecurity tasks.

Supervised ML and signature-based methods work well with things that are stationary. An ML algorithm trying to identify pictures of cats, will still be able to do so till the time cats start to look very different. In the domain of cybersecurity, active adversaries are looking at what defences work best and are actively trying to break them. RL can prove to be useful for constantly changing domains like cybersecurity, since it internally creates a simulation map of different variants that may spin off from the data presented for training. This helps deal with the problem of new attacks used by adversaries. Our hypothesis is that exploratory approaches will work well in this scenario, and thus we choose RL because it can use this to its advantage. Since adversaries are continuously evolving, pure tacit knowledge may not be very useful always. A combination of explicit and tacit knowledge will work well.

Chapter 2: Background for RL and CKG

In this chapter, descriptions of some of the key elements of Reinforcement Learning and Knowledge Graph are provided.

2.1 Reinforcement Learning

Reinforcement Learning(RL) has been around for a significant period of time. The way reinforcement learning differs from other forms of machine learning is that it espouses a more 'natural' way of learning. We have an 'agent' that interacts with the 'environment'. The agent takes certain 'actions' and based on either the quality of the action, or the states that the agent ends up in by taking an action, a feedback is provided from the environment. This feedback is called the reward. The key elements of an RL system are as follows.

- 'State': A state is a particular position the agent is at during training. For example, if we are training an RL agent to play a game of chess, the state would be the distribution of all the chess pieces across the chess board, at a given instance.

- 'Action': An agent takes a particular 'action' in order to transition from a given state to another state. For example, if an agent is at a state s, and it takes a particular action *a*, it will end up in another state *s'*.

- 'Reward': The transition from one particular state to another is marked by a feedback. Every transition is associated by the quality of the transition based on how good the 'reached' state is, or how good a state the agent can reach from the 'reached' state. The reward is expressed in the form of *r(s,a,s')*. This is a function that has as parameters, the previous state, the action that was taken by the agent and the state reached.

- 'Environment': An RL environment is what dictates the state transfers after taking specific actions, as well as the reward distibution. The environment can be a model-based environment or a model-free environment. A model-based environment obeys a fixed probability distribution for state changes. In particular we are talking about the probability distribution $p(s', r(s, a, s')|s, a)$. This helps calculate rewards and future rewards better. However, this is unfeasible in different situations because the probability distribution may be unknown for state changes. For example, for a drone that is flying in uncharted territories, it is impossible to know beforehand the probability distribution of different actions and the possible states it will end up being in. A model-free environment is one where the knowledge of transition probabilities is not compulsory.

- 'Policy': A policy can be considered as a mapping of states and actions for a particular agent. For every particular state, the policy dictates the action that the agent is going to take at that state. A deterministic policy is a direct mapping of individual states and actions. A randomized policy, however has a probability distribution that dictates the actions that are needed to be taken from particular states. Typically, a policy is represented by $\pi(a|s)$. This function, π, if randomized, is a probability of taking an action a from state s. In a problem where we are using an RL algorithm to solve, the policy is optimized so that it produces the best overall reward. An optimal policy is typically represented by the function $\pi^*(a|s)$.

- 'Value Function': A value function represents the quality of a state or a state, action pair. The value function *V(s)* represents the quality of the state s by aggregating the rewards and the future rewards that the agent is expected to collect from all the future states that can be reached from the current state. For a model-based RL system, the transition probabilities are already known. Hence, a single *V(s)* can be used to compute all the future rewards. As we can see in Equation 2.1, $V^{\pi}(s)$ represents the value function of state s following the policy π. Here γ represents the discount factor. The discount factor is a value between 0 and 1, and it represents how much we care about the future rewards as opposed to the immediate reward. A value closer to 1, places a higher importance on the future rewards. For a model-based system the transition probabilities are already known, and hence we can use a function called a transition function *T(s,a,s')* to calculate the value function.

$$V^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid s_0 = s, \pi\right] \tag{2.1}$$

$$V^{\pi}(s) = R(s) + \sum_{s'} T(s, a, s')\gamma V^{\pi}(s') \tag{2.2}$$

We have seen how we calculate value functions in a model-based setting. Next we are going to see how we utilize these methods in model-free reinforcement learning [16]. 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 [16].

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 [17].

$$\underbrace{\text{New}Q(s,a)}_{\text{New Q-Value}} = Q(s,a) + \underbrace{\alpha}_{\text{Learning Rate}} [\underbrace{R(s,a)}_{\text{Reward}} + \underbrace{\gamma}_{\text{Discount rate}} \overbrace{\max Q'(s',a')}^{\substack{\text{Maximum predicted reward, given} \\ \text{new state and all possible actions}}} - Q(s,a)] \tag{2.3}$$

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 [17].

Some other aspects of RL that will be of interest to us in the following sections are as follows:

- 'Deep Q Networks': Q-Learning is a great way of quantifying state-action pairs. The traditional way of computing Q-values is with the help of dynamic programming. However, they do not scale well. The complexity of Q-learning is $S.A$ where S is the cardinality of the number of states and A is the cardinality of the number of actions. As the number of states and actions grow to the range of millions, it becomes infeasible to calculate the Q-values iteratively. Deep Q-Networks, or DQNs are a great way of approximating the Q-values with the help of a neural network. Mnih et al. [18] put forward the loss function to approximate Q-values for a given state-action pair. The following equation tells us, how the Equation 2.3 can be used to create a loss function for DQN models.

$$\begin{aligned} L(\theta) &= E_{(s,a,r,s')}\left[\frac{1}{2}(\text{Bellman} - Q(s,a;\theta))^2\right] \\ &= E_{(s,a,r,s')}\left[\frac{1}{2}\left(R(s,a,s') + \gamma \max_{a'} Q(s',a';\theta) - Q(s,a;\theta)\right)^2\right] \end{aligned} \tag{2.4}$$

The policy gradients can be calculated as such:

$$\begin{aligned} \theta_{s,a} &= \theta_{s,a} - \alpha \frac{\partial}{\partial \theta_{s,a}} L(\theta) \\ &= \theta_{s,a} - \alpha \left(R(s,a,s') + \gamma \max_{a'} Q(s',a';\theta) - \theta_{s,a} \right)(-1) \\ &= (1-\alpha)\theta_{s,a} + \alpha \left(R(s,a,s') + \gamma \max_{a'} Q(s',a';\theta) \right) \end{aligned} \tag{2.5}$$

- 'Inverse RL': In the situations described above, we are working on an assumption that we know the reward distribution beforehand. This may not be the case all the time, as in domains such as cybersecurity, we may not have the knowledge for being able to quantify the good and the bad states. To tackle these problems, Inverse RL was introduced [19]. Here, the RL agent learns from an expert performing the task. The goal of this algorithm is to model ϕ in such a way that the policy emulates the expert. By emulating the expert, the algorithm learns how to assign rewards for different state-action pairs. Here μ_E is the feature estimate of the expert.

$$\begin{aligned} R(s,a,s') &= \phi(s) \\ minimize(\phi\mu(\pi^*) &- \phi\mu_E) \end{aligned} \tag{2.6}$$

2.2 Cybersecurity Knowledge Graphs

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. The classes and possible relationships between their instances are dictated by a pre-defined schema.

This schema also helps in reasoning, as it can help define specific constraints and axioms that may exist between the classes. For example, we can define a class ‘A’ disjoint from a class ‘B’. So, if we assert an entity in class ‘A’ and ‘B’ both, the knowledge graph reasoner is going to display an error and not let us perform the action.

Knowledge graphs for cybersecurity have been used before to represent various entities [20]. Open source CTI has been used to build CKGs and other agents to aid cybersecurity analysts working in an organization [21–25]. CKGs have also been used to compare different malware by Liu et al. [26]. Some behavioral aspects have also been incorporated in CKGs, where the authors used system call information [27]. Graph based methods have also been post processed by machine learning algorithms, as demonstrated by other approaches [28–30].

Syed et al. in their paper about the Unified Cybersecurity Ontology [31] proposed a schema to represent cyber-threat intelligence. UCO 1.0 was based on STIX 1.2 [32], which is a standard to share cyber-threat information. UCO 2.0 was based on STIX 2.0 [33], and it has refined the previous version of STIX [20]. We build the schema of our CKG based on the classes and relationships specified in UCO 2.0. Some important classes present in UCO 2.0 which are useful in our CKG, along with some additional new classes to better represent cyber-threat intelligence from cybersecurity text are:

- *Software* : An entity that relates to a piece of code usually used as tool such as Office or Adobe.

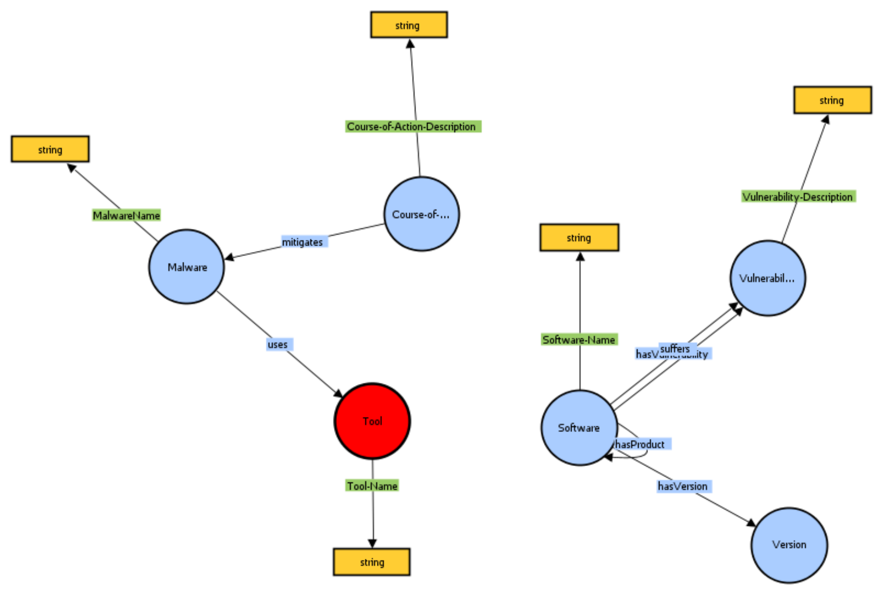


Figure 2.1: Some of the classes and the relationships as shown by the VOWL visualizer.

- *Exploit-Target* : An entity that relates to the site of the attack usually targeted by a malware such as Android or an operating system like Windows.
- *Malware* : An entity that refers to malicious code and/or software which is inserted into a system.
- *Indicator* : An entity that contains a pattern which helps the administrator to indicate an ongoing attack or malicious activity.
- *Vulnerability* : An entity that refers to a patch of bug or weakness that could be exploited by ill-intended users.
- *Course-of-action* : An entity that refers an action or set of actions that either prevents or responds to an attack.
- *Tool* : An entity that refers to legitimate software that can be used by threat

actors for malicious activities.

- *Attack-pattern* : An entity that refers to steps that could result in an active attack on an individual or group of users.

- *Campaign* : An entity that refers to grouping of activities that could lead to a malicious attack.

- *Filename* : A file which is used by the malicious software to execute the attack

- *Hash* : A SHA-256 hash of an executable which may be used to identify an attack

- *IP Addresses* : An IP Address or addresses which a malicious software may be using in the course of the attack

Pingle et. al, in their paper about CKG improvement and UCO 2.0 [20], have defined the classes and the relationships, necessary for the schema of our CKG.

Chapter 3: Related Work

In this chapter, we describe some of the relevant work that has been done in this research area.

3.1 OSINT and CKG

Unstructured text about a cyber-attack provides security analysts with knowledge, extracted and captured in a mandated format, so that they are able to use this information to analyze future attacks. The vast collection of these text reports about cyber-attacks fall under the umbrella of Open-Source Intelligence or OSINT.

There is a wide variety of OSINT that is available to researchers. Although they have varying degrees of credibility, all these sources are capable of providing critical information that can help security researchers identify cyber-threats. There are organizations like CVE [34]that collect Cyber-threat Intelligence (CTI). The cyber-threat information contains textual descriptions of known vulnerabilities. Other sources of similar information are the National Vulnerabilities Database [35] and CWE [36]. Technical reports published by official security agency websites like Kaspersky [37] and FireEye [38] make a reliable source of information to mine intelligence. This makes these reports a credible source of intelligence, as opposed to mining data from dark web logs, or social media as demonstrated by Mittal et al. [21–23], where the authenticity of the information, cannot be verified. However, even dark web logs, technical blogs, and social media contribute to the cyber-knowledge available to the general public.

After Action Reports (AARs) are different from technical blogs and include more technical details about malware behaviour. An AAR gives security analysts technical details and knowledge about a cyber incident. Cybersecurity blogs, differ from AARs and do not capture ample technical information. Sometimes, even superficial information about cybersecurity can be helpful for modeling cyber-threats and representing them. Apart from these sources, there are software companies that publish security vulnerabilities existing in their own products [39, 40]. They often mention possible ways to patch those vulnerabilities which also makes beneficial sources of information for aggregating security intelligence.

3.1.1 Cybersecurity Knowledge Graphs

A knowledge graph is a set of semantic triples, which are pairs of 'entities' with 'relationships' between them. Cybersecurity Knowledge Graphs (CKGs) have long been used to represent Cyber Threat Intelligence (CTI). To represent CTI in a CKG, the first step is to identify what entities and relationships need to be asserted. We also use an ontology called 'Unified Cybersecurity Ontology' (UCO) [31] to provide our system with cybersecurity domain knowledge. UCO is based on Structured Threat Intelligence Language (STIX 2.0) [33] which provides a schema to represent cyber-threat intelligence. CKGs have also been developed from other open-source information by Mittal et al. [21, 23]. CKGs and knowledge graphs have also been used to create various analyst augmentation systems [24, 25, 29, 30, 41, 42]. Next, we discuss the 3 main components of our system - a named entity recognizer, a relationship extractor, and a system to compare the various malware nodes in the CKG.

3.1.2 Named Entity Recognition

Named Entity Recognition (NER) for cyber-threat information extraction have been built using Conditional Random Fields (CRF), Support Vector Machines [43], and neural networks [44]. Ekbal et al. in their paper [43] have proposed a language independent algorithm for detecting named entities. Recently, Bi-directional LSTMs are being used to recognize named entities. Even in the field of CTI, entity extraction has been done with the help of deep learning [45] [46]. Some of the approaches, in the field of CTI, have also demonstrated the use of neural networks on hand-picked features which yielded better results [47] [48]. Despite the widespread applications of Long Short Term Memory (LSTM), the use of CRF-based classifiers for entity extraction have continued to remain state of the art [49]. Using BiDirectional LSTMs, aids in the process of capturing (or forgetting) long term context, which is necessary to predict an entity class. However, some of the entities that we are interested in, like 'filenames', 'IPAddresses', or 'hashes', do not need contextual information to be predicted into the correct class. Capturing diverse context for these classes may even be detrimental for the task of entity classification. Moreover, BiLSTMs overfit with limited data. Thus, we build our own entity extractor for cybersecurity text based on CRFs and regular expressions (See Section 5.1.1).

3.1.3 Relationship Extractor

Relationship extraction predicts the links or the relationships existing between pairs of entities extracted by our system. There has been significant work done in the field of relation extraction between entities. Relationship mapping can be many-to-many, many-to-one, or one-to-one. TransH models [50] have worked on extracting many-to-many mapping by shifting vector spaces in hyperplanes. TransE models [51] have used head and tail entities to predict one-to-one mapping. Sparse vectors have also been used [52] to predict relationship mapping, which was an improvement over TransH and TransE models. We use the Relationship Extraction method proposed by Pingle et al. [20], which is our previous work in this area. The Relationship Extraction algorithm uses vector encodings of individual entities created using word2vec [53]. These vector representations help capture the context of the cybersecurity entities in text, which aids in the task of relationship prediction.

3.1.4 Utilizing CKG for Malware Comparison

There has been significant research done in the area of comparing malwares. Some of these approaches use machine learning after extracting features about these malwares [26]. One interesting approach generates graphs on instruction traces of target executables [28]. The authors subsequently used machine learning algorithms to classify various softwares as benign or malicious. Other graph-based approaches include building behavioral graphs of malwares, based on system calls as demonstrated by Park et al. [27]. After constructing these graphs, the authors propose a method of subgraph matching to calculate similarity between malwares. However, this approach would encapsulate only the system behaviour at the site of attack. It does not necessarily capture broader details like, what is the target software, or if this malware is a part of a bigger campaign. It will not capture, the attack pattern of the malware in natural language. The CKG that we are building has technical details, as well as broader details about campaigns launched, tools used, softwares targeted, etc. Thus we can use the CKG, extract triples about a malware, and compare it with the triples about another malware and simply compare them, to cluster similar malwares. Jiang et al. in their paper [54], have proposed a method of recreating the semantic view of the host machine in a virtual machine and running malware analysis in the VM to tackle the problem of malware hiding from detection software. The paper demonstrates how, by recreating the semantic view of the host machine in the VM, it is possible to identify 'self-hiding' malware by using file comparisons. Although it is possible to detect newer varieties of malware using the semantic view, it does not give us higher level details about the malware. The semantic view that has been recreated in the paper is focused on system specific metrics. Some examples of these metrics are processes, memory, files. Our CKG based on STIX, captures not only system specific information, like filenames and hashes, it is also able to capture a wide range of details which aids security researchers. For example, by performing analysis in an infected machine, we may be able to gather the information that a file is suspiciously trying to connect to a remote IP Address and is trying to transmit some information. AARs, having already analyzed this may be able to assert that this is due to the 'command and control infrastructure' that the malware is using. Since we gather our information from AARs, we identify these keywords and populate the CKG. So a security researcher can easily search for all malwares using 'command and control' infrastructure in our CKG and will be presented with appropriate results.

3.2 Reinforcement Learning in Cybersecurity

Security is critical in maintaining the integrity of cyber systems. Technological innovations have lead to complex system architectures that are increasingly hard to protect. Pre-emptively 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 [55], malware detection [56], and cyberphysical attacks [57]. 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 1.1.

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 [58], robotics [59] and even for biological data [60]. 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. [61] 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. [62] 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. [63] 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. [64] discuss the challenges in detecting malicious behavior in Wireless Sensor Networks (WSNs). They explain why traditional intrusion detection methods fail 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. [65] 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 exploration 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.

Chapter 4: Methodology

4.1 Main Architecture

The system that we are proposing will have two main components. One significant component is building a cyber knowledge extraction pipeline that can take an unstructured piece of text describing a cyber-attack as an input and identify key pieces of information present in it. We have presented a brief description of cyber-knowledge extraction in general in Section 2.2. The second component is using this information in a RL algorithm for different cybersecurity tasks like malware detection, mitigation, and also adversarial malware generation.

If we look at Figure 6.5, we can see how an unstructured piece of text is provided as an input to the knowledge extraction pipeline. The unstructured piece of text can be a malware After Action Report, social media feeds, dark web or reddit logs about malware, etc. It is then fused with other sources of information that can be structured or unstructured. It will be helpful if these sources of information can tell us about how certain system parameters behave in the presence of a malware, compared to the absence of it. This enriched knowledge is particularly useful for detecting new variants of malware. By querying this CKG that has been enriched by the fusion process, we can retrieve key pieces of information that can help us model an RL algorithm for specific cybersecurity tasks. We will first talk about the motivation for selecting our tasks and then move onto more details of our individual components.

4.2 Motivation for Task Selection

From the perspective of a cybersecurity professional, it helps to visualize cybersecurity problems with three categories.

- 'Nothing bad will happen. Find out before it acts'
- 'Something bad has happened. How do we stop it?'
- 'Something bad will happen. How do we know beforehand?'

The primary work that is going to help us in achieving all the aforementioned tasks is automating the interaction between a CKG and the RL algorithm designed specifically for the task. The parameters of the RL algorithm will change as the tasks change, and we want to retrieve those parameters from a CKG instead of handcrafting generic rules that may not perform well for all tasks.

When we speak of finding out before a malware acts, we mean improving the detection of the malware. This can also be mapped to mitigation tasks, if we are patching vulnerabilities before an adversary has gained access to a system. When we say that 'something bad has happened', we mean an adversary has already gained access to a victim's system. The goal here should be to stop the adversary from gaining access to all the sensitive details of the system. We can visualize this particular situation as a defender working against an adversary in a 'Capture the Flag' type setting. The last category talks about knowing beforehand that something bad will happen. This can be mapped to adversaries that can also leverage the information present in CKGs to generate more efficient malware. If we create a model that can use the CKG's information, we will be able to include these malware samples in a potential malware classifier in future, or even in our malware detection framework. As we said before, we are now going to discuss the components of our proposed system that can help us achieve these tasks.

4.3 Cyber Knowledge Extraction

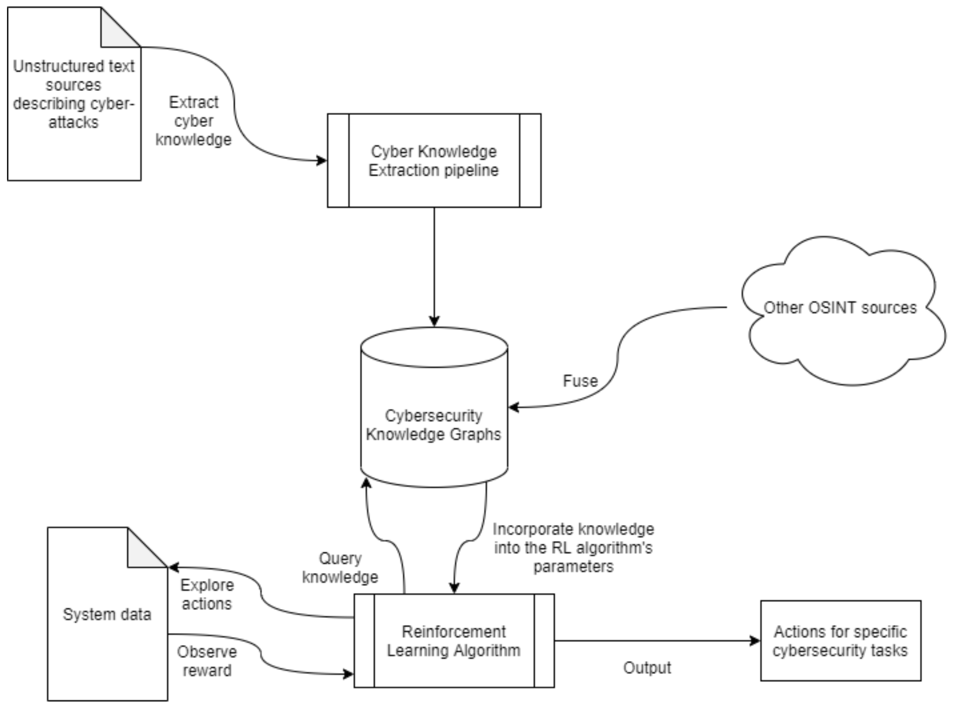

Unstructured text sources describing cyber-attacks
Extract cyber knowledge
Cyber Knowledge Extraction pipeline
Other OSINT sources
Fuse
Cybersecurity Knowledge Graphs
Query knowledge
Incorporate knowledge into the RL algorithm's parameters
System data
Explore actions
Observe reward
Reinforcement Learning Algorithm
Output
Actions for specific cybersecurity tasks

Figure 4.1: An architecture diagram specifying the different steps of our method

The cyber knowledge extraction pipeline, as seen in Figure 4.1, takes an unstructured piece of text as an input and produces semantic triples relating to cybersecurity. Security researchers perform analysis on malware samples and publish them in the form of technical reports or After Action Reports. Some of this information can also come from companies that are actively involved in finding security threats in their own products, and benevolently informing the general public of their findings [39,40]. The knowledge extraction would require state-of-the art NLP techniques to identify key entities present in an unstructured piece of text about cybersecurity. The first part of the knowledge extraction would relate to this. The key entities that are extracted from the piece of text requires Named Entity Recognition. By recalling the useful classes and relationships that we mentioned in Section 2.2, we are able to identify the target classes. In order to train a classifier that is going to extract information that can be mapped to those classes, we need an annotated corpus of cybersecurity text. We want this classifier to work on a variety of cybersecurity text. Since long pieces of cybersecurity text and smaller pieces of text like CVE descriptors, that are also useful, are structurally very different, our corpus needs to be inclusive of examples from a variety of sources. The trained classifier for Named Entity Recognition needs to extract entities from these sources and also other pieces of unseen text.

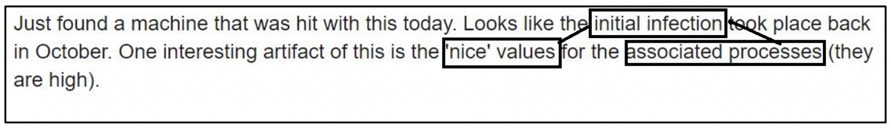

Just found a machine that was hit with this today. Looks like the initial infection took place back in October. One interesting artifact of this is the 'nice' values for the associated processes (they are high).

Figure 4.2: An example of extraction of entities and relationship extraction from a small piece of text

The output of the previous step is a list of entities extracted from a piece of text. The next step in this process is relationship extraction. We provide pairs of extracted entities as an input to the relationship extractor and we expect a correctly predicted relationship that should exist between the pair of entities. If there is no possible relationship between the pairs of entities, the relationship extractor should be able to identify that as well. Figure 4.2 provides an example of how the knowledge extraction would work on a small piece of text.

4.4 Enrichment of CKG by Fusion of Knowledge from Other Sources

The CKG should not limit itself to the knowledge extracted from text sources. The knowledge does not always come from credible sources, since we plan on using open-source text. In order to improve this knowledge, we also need information resulting from the actual detonation of a malware.

This not only improves the quality of knowledge, but it also enriches our CKG as it provides the CKG with additional knowledge. One way to approach this would be to assert the behavioral data after detonation of a malware. If we get data about how certain system parameters change after a malware is active in the system, it would help an RL agent to observe those specific parameters to identify the presence of the malware.

4.5 Making CKGs Robust against Poisoning Attacks

After we create this knowledge graph, we need to ensure that the information present was correct and/or stood the test of time. Data poisoning is a real problem for these knowledge graphs built from OSINT. We create SVM-based ML models that detect poisoned data based on metadata from the data source, such as reddit text descriptors. This helped in processing the OSINT that is not poisoned. However, with the advent of Generative Pre-trained Transformers (GPT), it is possible to generate realistic OSINT for cyber-attacks. Detecting what is generated or fake CTI and what is real is challenging even for human experts. We create a graph neural-network-based trust score for semantic triples in our knowledge graph. This score can also be used to filter possibly poisoned semantic knowledge. We will discuss more about this in Sections 5.2 and 5.2.1.

4.6 Reinforcement Learning Tasks

We plan to address the issues mentioned in Section 4.2 with the help of three specific tasks. The first task, described in more detail in Section 6.2, helps in the interaction between RL and CKG for malware detection and mitigation. This follows the problem statement: ‘Nothing bad will happen’ and ‘Something bad has happened’. This interaction, if automated, can help the RL agent to communicate with the CKG to improve reward metrics and also explore more on some suggested actions. For detection and mitigation, an important step for RL is to identify the ‘state-action’ pairs that are good (or bad) for the objective. Our goal is to take advantage of the enriched CKG that contains information regarding the system parameters to look out for. These system parameters when incorporated in the reward function, can help in differentiating between the ‘good’ state-action pairs and the ‘bad’ state-action pairs. Another important step is to guide the exploration of these actions that are useful. An RL agent can take a long time to understand the actions that are proving to be useful for a specific task. Depending upon the problem and the RL agent’s exploration, it may or may not be able to find out the optimal action sequence for an objective. We can use the CKG to suggest certain actions to the RL agent that it can explore more, instead of randomly choosing an action to explore. This can help the RL agent to be successful in finding out the optimal action sequence. However, we do not want to limit the RL agent’s exploration to the suggested action. We want the RL agent to explore the suggested actions, as well as other actions through random exploration.

Further discussion on ‘Something bad has happened’ is presented in Section 6.4. Since CKGs contain information about the ‘Course-of-Action’ required to mitigate a threat, this becomes particularly useful for automatically generating strategies to thwart a cyber-attack. In order to anticipate future attacks and address the issue: ‘Something bad will happen’, we also discuss simulating a cyber-attack in specific environments to understand how certain nodes can have vulnerabilities exposed for an intelligent attacker. We leverage the information present in the CKGs to aid in the discovery of new attacker policies and defender policies. This is motivated by the use of both tacit and implicit knowledge in human intelligence in our day-to-day lives. The information contained in knowledge graphs can be imagined as absolute truth, which does not need to be learned (again) through experience. However, because these knowledge graphs often work in open-world domains, there is information that is true but not contained in these knowledge graphs. New information can be uncovered using exploratory algorithms, such as RL.

In the next sections, we will look at each of these components in more details. We will discuss our findings, and provide an outline of future directions of this research.

Chapter 5: Cyber Knowledge Extraction

In this chapter, we will discuss the core components of our model. We will discuss the experiments that we performed and our conclusions from them.

5.1 Named Entity Recognition (NER) and Relationship Extraction

In our paper [66], we described a system that takes After Action Reports (AARs) as an input and automatically generates a CKG as an ouput. The three main components of our system are as follows:

- ***MEE*** : A Malware Entity Extractor which has been trained over annotated pieces of cybersecurity text, to predict cybersecurity entities in an AAR.
- ***RelExt*** : A relationship extractor which has been trained over cybersecurity data, to predict a relationship between pairs of entities extracted by MEE.
- ***CKG*** : Cybersecurity Knowledge Graphs which are populated with the entity relationship sets of the extracted data. The CKG represents the unstructured data present in the AAR into a structured ontology.

We have published two key papers in this area [20, 66]. The MEE is a CRF based classifier that takes an unstructured text as an input and tries to map it to the UCO based classes mentioned in Section 2.2. For some of the classes the contextual information is not necessary, and the predictions are better if we simply concentrate on the structure of the words in the text. This is pertinent to ‘IP Addresses’, ‘Filenames’, and‘SHA-256 Hashes’. For these classes, we exclusively use regular expressions for detection. The Relationship extractor [20] is a deep neural network based classifier. It takes Word2Vec [53] embeddings of the pairs of entities extracted by the MEE as an input. The output is a relationship that exists between the pair of entities. The classification scores for MEE and RelExt can be seen in Tables 5.1 and 5.3 respectively .

5.1.1 MEE

General purpose NERs do not necessarily work well for a particular domain. In some cases, it is imperative that we build a dataset on our own if we want to build an NER for a particular domain. Most of the prior work on NER for cybersecurity has concentrated on CVE and NVD feeds for their data. Annotated datasets required for supervised learning are not readily available for cybersecurity. Even if an annotated dataset is available, it is difficult to reach a consensus about what classes need to be extracted from the data. Recently an NER corpus for cybersecurity has become available for researchers to use [67]. But even this dataset has concentrated only on twitter feeds, and annotated the feeds with a few classes that may not necessarily help cyber-researchers to extract meaningful information. In order to compare different NER algorithms, we need to test them on a dataset that has a mix of different types of feeds. We create a dataset of 2100 sentences collected from various sources.

- Microsoft Security Bulletin [39]
- Adobe Security Updates [40]
- Twitter feeds
- Long Technical Reports
- Common Vulnerabilities Enumeration [34]
- National Vulnerabilities Database [35]

These sources provide a mixed bag of sentences which is ideal for testing out different algorithms for NER. Some NER algorithms trained on CVE and NVD sentences may not work well for long technical reports. The information in sentences from small pieces of text is often condensed, compared to sparse descriptive reports about cyber-attacks. We have curated 550 pieces of text from Microsoft and Adobe and we have 474 Technical Reports about cybersecurity.

In Table 5.1, we look at the performance of a CRF based classifier on various cyber-entity classes. We also conduct a comparative study of deep learning algorithms for cyber entity extraction [68].

In Table 5.2, we observe that the models using BERT embeddings, even though not trained entirely on a cybersecurity corpus, achieve better scores than models using Word2Vec embeddings. Among the models using Word2Vec embeddings, we can see that the CNN based models perform slightly better than the CRF based

Entity Classes	Precision	Recall	F-1 Score
Attack-Pattern	77	63	69
Campaign	92	74	82
Course-of-Action	80	62	70
IP Addresses	100	88	93
Hashes	100	100	100
Exploit-Target	77	90	83
Filenames	80	90	85
Malware	83	83	83
Software	86	88	87
Tool	80	90	85
Vulnerability	73	53	62
Average	91	92	91

Table 5.1: Precision, Recall, and F-1 score for Entity Classes evaluated across the test sets of the corpus.

Table 5.2: Comparison of Results of Deep Learning Algorithms for Cyber Entity Extraction

Test and Validation Results					
Method	**Test Acc.**	**Validation Acc.**	**Precision.**	**Recall.**	**F-1.**
Stanford NER	91.6	90.8	78.00	78.00	77.00
Domain independent Word2vec+LSTM +CRF	96.96	96.71	85.00	77.00	81.00
Domain independent Word2vec+BiLSTM+CRF	97.40	97.09	88.00	80.00	84.10
Domain independent Word2vec+CNN+LSTM	97.59	96.64	84.00	80.00	82.00
Domain-specific Word2vec+LSTM +CRF	97.31	96.16	90.00	76.00	82.40
Domain-specific Word2vec+BiLSTM+CRF	97.20	96.80	88.00	81.00	84.30
Domain-specific+CNN+LSTM	97.66	97.18	90.00	81.00	85.20
BERT+LSTM+CRF	97.73	96.93	90.00	83.00	86.20
BERT+BiLSTM+CRF	98.10	97.18	93.00	84.60	88.60
BERT+CNN+LSTM	97.50	96.80	91.00	80.30	85.30

[a]F-1 score is the key evaluator for performance

models. The character level understanding is helpful for Word2vec, but since BERT has built-in character level information in its embedding step, it is not necessarily an advantage.

5.1.2 RelExt

We take pairs of malware entities captured by our MEE and pass it to the next stage of our pipeline, which establishes a relationship existing between a given pair of entities.The RelExt is essentially a neural network that takes two vectors representing two entities extracted by our MEE and predicts a relationship between

Relationship Classes	Precision	Recall	F-1 Score
hasProduct	49	97	65
hasVulnerability	92	74	82
uses	100	88	93
indicates	80	90	85
mitigates	55	70	62
related-to	92	74	66

Table 5.3: Precision, Recall, and F-1 score for relationship classes.

them. We train a Word2vec [53] model separately and the extracted entities are passed to the model to generate their respective embeddings. The dimension of the embeddings is 200. The RelExt neural network takes a pair of embeddings as input. It has ab input layer, 3 hidden layers, and a softmax layer as output. The dimension of the input layer is 400 and the hidden layers have dimensions 200, 100, and 50 respectively. The output softmax layer has a dimension of 6.

Before we send entity pairs to RelExt, we perform post-processing of candidate entity pairs based on the pre-defined schema of our CKG. We only pass the entity pairs, which can have a credible relationship between them, and we discard the pairs of entities which, according to the schema of our CKG, cannot have any relationship. For example, there can be no direct relationship between a 'Software' and a 'Filename', according to our CKG, so we do not pass entity-pairs which are of the type ‘Software’ and ‘Filename’ as candidates to our next stage. Since we have already defined a schema for our CKG, we automatically filter out pairs of entities which do not have credible relationships between them. This component of our pipeline, is a neural network, which takes in two entities as input and then predicts a particular relationship which exists between the two entities.

The first stage for relationship extraction is to represent the input entities in a vector form, which is suitable for neural networks to perform matrix multiplication. The output of the MEE (See Section 5.1.1), merely produces a class type, or entity type, for a particular word, which is not sufficient for neural networks. So we take the corpus we used for training the MEE, and generate word2vec embeddings [53] for each word. Word2Vec is a technique to represent words present in a corpus. Word2vec algorithm captures the context of each word in the corpus, and represents each word with a vector of a specified length. We train word2vec on a corpus of cybersecurity text from NVD [35], CVE [34], and STIX data from TAXII servers [20], to represent each entity captured by our MEE.

The second part of the relationship extractor is to train the neural network with the help of a training set made from NVD, CVE and STIX datasets. We have labeled pairs of entities and the relationship which exists between them. Our dataset consists of a total of 33,000 labeled relationships. We split the dataset into a training set and validation set. We use different split ratios: 80 (train), 20 (validation); 70 (train), 30 (validation); 90 (train), 10 (validation).

We then train the neural network model RelExt to produce the output class of relationship that exists between the pairs of embeddings. We see the performance of this relext model in Table 5.3.

5.1.3 CKG Assertion and Fusion

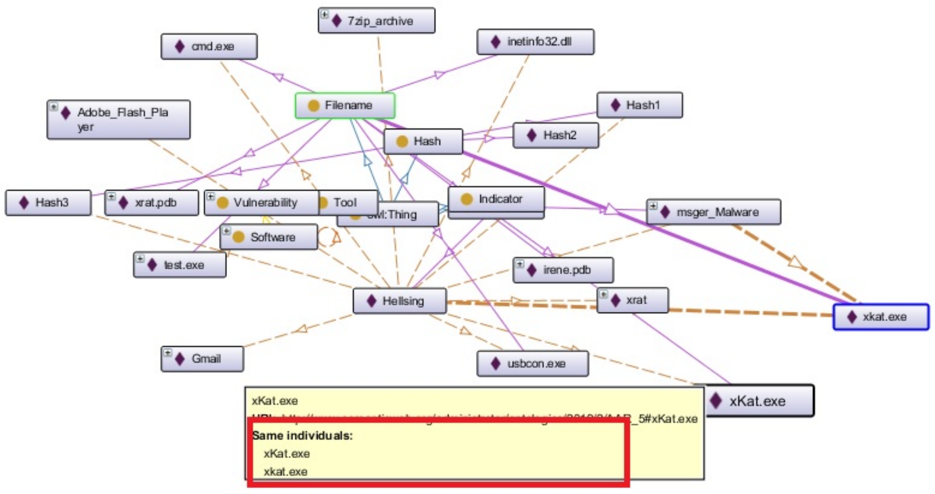


Figure 5.1: Hellsing malware entities in the Cybersecurity Knowledge Graph.

After we generate the entity-relationship set, which is the output of the relationship extraction component of our system, we are able to assert this data to our CKG. We base our knowledge graph schema on STIX (Structured Threat Intelligence) [33] and use UCO 2.0 [31] to provide cybersecurity domain knowledge to the system. The output of the relation extraction is a set of semantic triples and the types of the individual entities. Entity classes, relations, and the specific classes which act as a domain or range of each relation describe the schema of a knowledge graph.

In our dataset, we encounter multiple AARs describing the same attacks or malwares. Ideally, we would like to fuse knowledge extracted from different AARs describing the same malware to create a more robust CKG.

If an AAR entity has been already asserted in our CKG and an exact match is available, we simply declare a '*owl:SameAs*' assertion and fuse the graph entities. Here we declare that the two entities nodes and subgraphs are the same. If there is no exact match for a newly discovered AAR entity, we calculate the term frequency–inverse document frequency (TF-IDF) scores to calculate similarity between the current document and previously processed AARs. We also calculate string similarity between new entities and entities which are already asserted using various 'edit-distance' metrics. If there is a close match, we fuse them. Fusion helps us discover knowledge about a malware or a cyber-entity which is found in multiple AAR sources. This makes our CKG more robust, with more information about cyber-entities ingested. Here is an example query on the 'unfused' CKG.

```
 SELECT ?x WHERE {
  ?x a CKG:Malware;
     CKG:uses
      CKG:588f41bbc117346355113f.}
```

The above query returns:

```
Hellsing
```

The same query on the 'fused' CKG returns:

```
 SELECT ?x WHERE {
  ?x a FusedCKG:Malware;
     FusedCKG:uses
      FusedCKG:588f41bbc117346355113f.}
```

The above query returns:

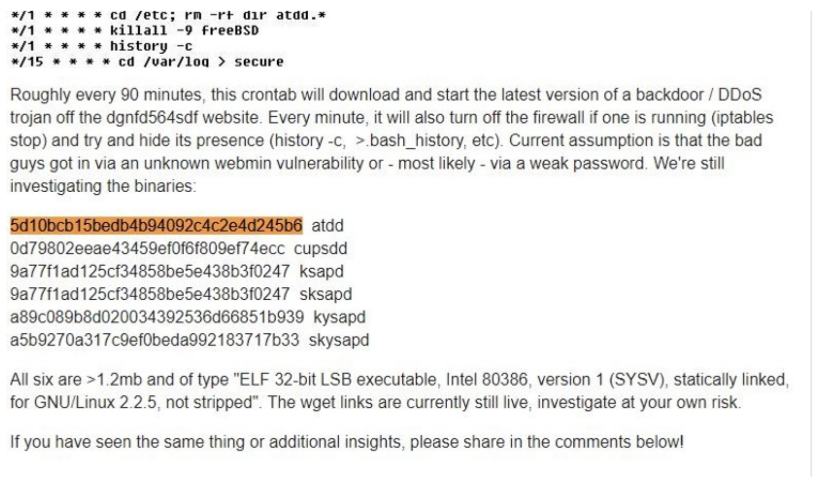
```
Hellsing, XWeber
```

In Figure 5.1 we see how the *Hellsing* malware and its attributes are identified and asserted in the CKG. We can see that ‘Hellsing’ ‘uses’ different files like ‘cmd.exe’, ‘test.exe’, and ‘xKat.exe’. This was created using the entity relationship sets from two reports about the same malware. One of them mentioned the filename ‘xkat.exe’ and the other report mentioned a filename called ‘xKat.exe’. **An e**dit-distance algorithm helped us calculate the similarity between the two entities and we used **an** ‘owl:SameAs’ **relation** to assert that those two individuals **were identical**. All relationships held by one of the nodes **will hold for** the other as **a result of the** ‘owl:SameAs’ assertion. We also see other entities identified. For example, ‘7zip archive’ is a ‘Tool’ which is used by the Hellsing malware.

5.2 Understanding Poisoning Attacks on CKGs and Knowledge Verification

A CKG that has been built from the information extracted from OSINT text is limited to the information provided by the author of the text. However, an author sometimes focuses on a particular aspect of the malware. For example, the writer of this text at Figure 5.2 does not say how she suspects that the firewall has been compromised nor how she determined that the file was being downloaded every 90 minutes. However, she does infer that the malware has been using a ‘weak password’ vulnerability, which is something that may not be easily inferred by simply observing

(a)



```
*/1 * * * * cd /etc; rm -rf dir atdd.*
*/1 * * * * killall -9 freeBSD
*/1 * * * * history -c
*/15 * * * * cd /var/log > secure
```

Roughly every 90 minutes, this crontab will download and start the latest version of a backdoor / DDoS trojan off the dgnfd564sdf website. Every minute, it will also turn off the firewall if one is running (iptables stop) and try and hide its presence (history -c, >.bash_history, etc). Current assumption is that the bad guys got in via an unknown webmin vulnerability or - most likely - via a weak password. We're still investigating the binaries:

```
5d10bcb15bedb4b94092c4c2e4d245b6 atdd
0d79802eeae43459ef0f6f809ef74ecc cupsdd
9a77f1ad125cf34858be5e438b3f0247 ksapd
9a77f1ad125cf34858be5e438b3f0247 sksapd
a89c089b8d020034392536d66851b939 kysapd
a5b9270a317c9ef0beda992183717b33 skysapd
```

All six are >1.2mb and of type "ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), statically linked, for GNU/Linux 2.2.5, not stripped". The wget links are currently still live, investigate at your own risk.

If you have seen the same thing or additional insights, please share in the comments below!

(b)

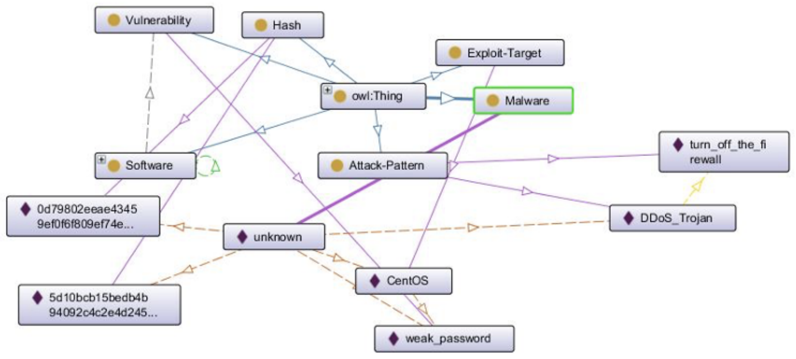


Figure 5.2: (a) An excerpt of a CTI describing a malware. (b) The knowledge graph representation of the CTI.

the system parameters of the behavior data. If we merge the information that has been stated online about the malware with the behavior data, we can get an *enriched* CKG.

If we only consider information extracted from open-source text, our CKG will have to rely on the conclusions derived by the authors of the text. For text written by a trusted security organization, we can say with some confidence that the information is likely to be correct. However, very often, open source text describing a malware is written by someone whose identity cannot be verified, which raises a question on the quality of information present in the knowledge graph. Since our knowledge graph's underlying schemas are based on ontologies, constraints, and rules defined in OWL [69], contradictory information can be detected. If an author claims that the network activity will increase after the malware infects a system, and our behavior data says that there has been no increase in the network activity, we can infer that there is a *reasoning error*. This will lead us to conclude that the information stated by the author may not be correct, thereby improving the reliability of our CKG. In this paper [41], we describe a method to collect the system behavior when a malware is active in the system. We analyze the system behavior and fuse this data with CTI collected from OSINT. If we want to query from the information asserted in the fused knowledge graph, we can run the following query. This simply translates to 'which parameters have their maximum values changed for the malware which has the hash: '5d10bcb15bedb4b94092c4c2e4d245b6'. A point to be noted here is that the values returned have their maximum values changed by a significant amount. For our experiments, we chose this threshold as 30%.

```
SELECT ?x WHERE {
  BKG:5d10bcb15bedb4b94092c4c2e4d245b6
     BKG:hasParameter ?x.
  ?x BKG:parameterchange
     BKG:increases_maxchange.}
```

This query results in two values:

```
cpu_children_sys,io_write_bytes
```

5.2.1 Fake CTI Generation

In this paper [70], we highlight the potential risks of fake Cyber Threat Intelligence (CTI) being used to attack cyber-defense systems. We demonstrate how transformers, such as GPT-2, can automatically generate plausible fake CTI text that can mislead these systems. The generated fake CTI can be used in data poisoning attacks, leading to incorrect reasoning outputs, and representation poisoning. The study reveals that cybersecurity professionals and threat hunters were equally likely to consider the generated fake CTI as true, emphasizing the need to address this issue.

Generation Input	Cyber Threat Intelligence	True/Fake
APT41 is a state-sponsored espionage group	APT41 is a state-sponsored espionage group which operates against higher education, travel services, and news/media firms which provide some indication that the group also tracks individuals and conducts surveillance.	True CTI
	APT41 is a state-sponsored espionage group. The Russian, Moscow-based group is responsible for capturing and stealing computer network communications and intellectual property. APT41 employs social engineering techniques to get around network defenses designed to last for years, and is capable of compromising and stealing dozens of companies	Fake CTI (generated)
Malicious Domain in SolarWinds Hack Turned into 'Killswitch'	Malicious Domain in SolarWinds Hack Turned into 'Killswitch'. A key malicious domain name used to control potentially thousands of computer systems compromised via the months-long breach at network monitoring software vendor SolarWinds was commandeered by security experts and used as a "killswitch" designed to turn the sprawling cybercrime operation against itself.	True CTI
	Malicious Domain in SolarWinds Hack Turned into Killswitch Service. This attack took place last year and affected hundreds of organizations. The attackers took advantage of the vulnerability in Win32k framework to attack their targets through crafted code.	Fake CTI (generated)
An issue was discovered in the Quiz and Survey Master plugin	An issue was discovered in the Quiz and Survey Master plugin before 7.0.1 for WordPress. It made it possible for unauthenticated attackers to upload arbitrary files and achieve remote code execution.	True CTI
	An issue was discovered in the Quiz and Survey Master plugin 1.2.1 for WordPress. By setting a CSRF token, an attacker can take advantage of a vulnerability in the survey.php admin panel.	Fake CTI (generated)

Table 5.4: Fake CTI Samples produced by our fine-tuned GPT-2 model.

In the Table 5.4, we observe some of the CTI text generated by the GPT-2 model. In comparison, we also see similar true CTI for reference.

		Participant Labels		
		True	False	Total
Actual Data	True	206	74	280
	False	220	60	280
	Total	426	134	

Table 5.5: Confusion matrix of participants evaluating CTI as true and false

In Table 5.5, we observe that human participants were able to label 206/280 actual true labels as true CTI, but when asked to label 280 fake CTI, they labeled 220 of them as true. Although the majority of the fake CTI contained entities (such as products and attack vectors) that were unrelated to each other, we found if the sentence structure displayed little or no linguistic deficiencies, the data was likely labeled as true. We also noticed sources that lacked substantial context were likely labeled as false. The generated fake CTI not only has the ability to mislead cyber-security professionals, but also has the ability to infiltrate cyber defense systems. In the next section, we describe how the generated fake CTI examples can be used to launch a data poisoning attack.

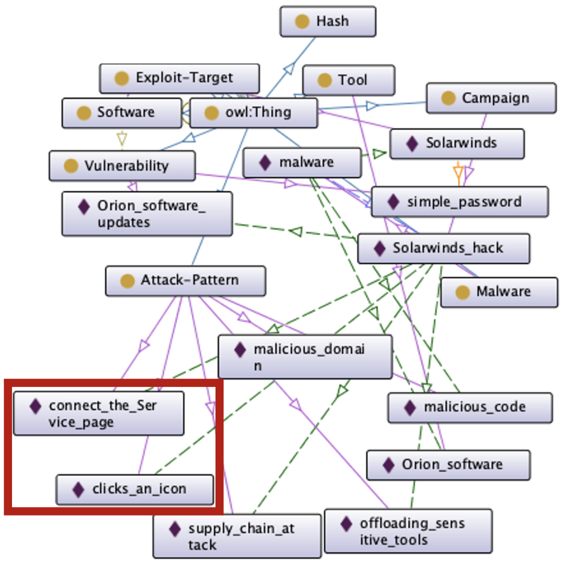


Figure 5.3: The poisoned CKG with additional data (red box) extracted from fake CTI.

5.2.2 KG Correction and Provenance

In this paper [71], we discuss a Graph Neural Network-based model that will help us generate scores for the specific relationships that exist between pairs of entities in a CKG. We use a supervised approach, where we use known triples that are correct, as the ‘labels’. The relationships for all the entities that do not belong in the ‘label’ are also updated simultaneously. Each relationship in the graph has a score, that gets updated after receiving the supervision from the ‘label’. We want to use this updated score to filter out the relationships that are possibly incorrect due to being outdated, or simply because they were from a data source that had incorrect information. This will help us preserve the relationships that are still correct, and discard the relationships that are not.

The problem statement of our paper is to find out scores between 0 and 1 to represent each relationship in a CKG. The higher the score the more credible or trustworthy the relationship is.

Let us consider that we have a set of entities E and a set of relationships R. Each semantic triple in the CKG is represented as $K = \{E_h, r, E_t\}$, where E_h represents the head-entity, E_t represents the tail-entity, and r represents the relationship between them, such that $r \in R$.

We represent each input CKG in the form of a matrix K, where

$K = \{0, 1\}^{|E|\times|E|\times|R|}$

Here $|E|$ denotes the number of entities in K and $|R|$ denotes the number of relationships. It is a symmetric matrix, where each row denotes if the particular relationship exists or not for the entity pair. For example if $K(i, j, k) = 1$, it means R_k exists between E_i and E_j. If it is 0, then the relationship R_k does not exist between entities E_i and E_j. After successful training, $K(i, :, :)$ can be interpreted as an embedding vector representing E_i that tells us how much credible all the relationships are with respect to all the other entities in the CKG, for the entity E_i. For example if a *Malware* 'Solarwinds hack' *uses* an *Attack-Pattern* 'clicks an icon', we have to index these entities 'Solarwinds hack', 'clicks an icon', and the relationship 'uses'. Supposedly, the indexes come out to be E_3, E_7, and R_2, respectively, then $K(3, 7, 2)$ becomes 1.

The input to the system is a CKG that is represented by K consisting of partially correct information. We use a CKG that holds the correct entities and relationships in place that we denote by K^*. The GCN is a function approximator,

f , that produces another CKG K' as output such that

$K' = [0,1]^{|E|\times|E|\times|R|}$ and

$K' = f(K)$

In this case, $K'(i,j,k)$ is a probabilistic value between 0 and 1 that tells us how important the relationship R_k is between entity-pairs E_i and E_j.

We use K^* as the target. As discussed before, each $K^*(i,:,:)$ gives us the ideal representation of entity E_i. The neural network f produces an output K' and $K'(i,:,:)$ represents the predicted representation of the same entity E_i. We define a loss function L such that it calculates the dissimilarity between the predicted representations of all the entities and the target representations of all the entities. The cumulative loss function is defined as follows.

$Loss = \sum_{i=1}^{|E|} L(K^*(i,:,:), K'(i,:,:))$

The construction of K^* does not require an entire CKG that has been rectified by human annotators. The purpose of our research is to update the probabilistic values that signify the 'trust' scores of the relationships of our CKG. In order to make our algorithm scale well, we cannot place a requirement on a completely correct CKG for training. If we do not have a complete CKG to be used as label, and we have partial triples that talk about the same malware we generate a CKG on our own in the following way.

For supervision, let us consider we have a set of triples T_p. In the original graph we have a set of triples T_s. Each triple $t \in T_p$ can be represented by P_h, P_r, P_t corresponding to the head-entity, relationship, and the tail-entity. Similarly, each triple $t \in T_s$ can be represented by S_h, S_r, S_t. We iterate through all the triples in

T_p, and check if the head of any triple in T_p overlaps with any of the tail nodes of the triples in T_s. If that node is $T_p(i)_h$, we remove all triples from T_s, where $T_p(i)_h$ is a head node. After that we add all triples from T_p to T_s, that have $T_p(i)_h$ as a head node. We remove them from T_p. We repeat this for all the remaining nodes in T_p untill it is empty. We discard nodes that have no overlap with $|E|$ which is the set of entities in K. The key motivation for doing so is that T_p can come from a source that talks about some specific details of a cyber-attack. We want to update the knowledge graph for those specific details of the cyber-attack, but we also want to preserve the semantic triples that talk about the other aspects of the cyber-attack that have been collected from other CTI sources that is represented by T_s. For example, consider a malware 'Dark Caracal' that uses another malware 'xRAT'. The entire CKG that has information related to 'Dark Caracal' and 'xRAT' is T_s. Suppose, a new report comes that has updated information about 'xRAT'. We want to create a new CKG that retains the information about 'Dark Caracal', but updates the information about 'xRAT'. A possible critique of this method could be removal of all knowledge about 'xRAT' from T_s that is not covered in T_p. It should be noted here that a missing triple would not mean that after training, the scores of the relationships belonging to the removed triples will be low. The GCN model changes the scores based on subgraphs and it is a self-correcting system. With more CTI being available about 'xRAT' it will be possible to change the scores of the relationships from the discarded triples, if they happen to be correct. We can see the algorithm in Algorithm 1.

It is imperative that we record key information about the source of the threat-

Algorithm 1 Algorithm to create K^* from partial sub-graphs

```
  $N_s \leftarrow |T_s|$
  $N_p \leftarrow |T_p|$
  **while** $N_p \neq 0$ **do**
    $P_h, P_r, P_t \leftarrow T_p(N_p)$
    **while** $N_s \neq 0$ **do**
      $S_h, S_r, S_t \leftarrow T_s(N_s)$
      $C \leftarrow \emptyset$
      $A \leftarrow \emptyset$
      **if** $P_h$ equals $S_t$ **then**
        $C \leftarrow T_s(i) \quad \forall i \in N_s \;\; s.t. \;\; T_s(i)_h = P_h$
        $A \leftarrow T_p(i) \quad \forall i \in N_p \;\; s.t. \;\; T_p(i)_h = P_h$
      **end if**
      $T_s \leftarrow T_s - C + A$
      $T_p \leftarrow T_p - A$
      $N_p \leftarrow |T_p| - 1$
    **end while**
    $N_s \leftarrow |T_s| - 1$
  **end while**
  **return** $T_s$
```

intelligence. For example, if a CTI originating from a Twitter post suggests that a cyber-threat can be mitigated by updating Adobe Acrobat, it should carry less weight than a CTI from Adobe Security Bulletin that says otherwise. Recording provenance level information provides us with the ability to create additional filters on the threat-intelligence that practitioners use.

In order to effectively use CTI to evaluate and mitigate risks, organizations need to process the raw data feeds and appropriately represent them. Cybersecurity Knowledge Graphs (CKG) are becoming popular to represent CTI. This is because CKG has reasoning capabilities that are applied on the interconnected entities. These reasoning capabilities enforce rules that help in preserving credible information and discarding the rest. Additionally, classes capturing the provenance can be added in the schema of the CKG that can give us more information about the source of the data. In our paper, we improve an existing CKG schema that records CTI with additional classes and relationships that indicate provenance.

The schema of the CKG, as mentioned before, tells us the entity-classes and the relationship-classes. After the schema is defined, various semantic triples are asserted in the CKG. The semantic triples record a specific relationship existing between pairs of entities. Unstructured sources for CTI need to be processed to extract the entities and relationships. We use Natural Language Processing (NLP) techniques to extract the CTI information representing a cyber-attack or a malware, along with the information related to the provenance. The objective of this paper [72] is to associate provenance with the entities asserted in the CKG.

The provenance system is to gather CKG sources to address the authenticity of that data. We consider URL, Publisher, etc. as a source. In Provenance Extractor (ExP), we extract triples mentioned in UCO 2.0 from the data source to draw relations among them using the Provenance Encoder.

Syed et al. [73] in their paper have developed the Unifed Cybersecurity Ontology (UCO) 1.0, which was updated to UCO 2.0 [20]. These updates in the ontology have been in line with the updates in STIX from 1.0 to 1.2 [33]. We extend UCO 2.0 to add provenance information into the CKG by redefining the existing schema and possible relations. Next, we explain various provenance classes that have been added in UCO 3.0, which inherits the cybersecurity schema and domain knowledge from UCO 2.0.

- *Intelligence-Id*: An entity class that refers to a unique CTI represented in our CKG.
- *URL*: An entity class that refers to the originating URL from where the CTI was collected.
- *Organization-Name*: An entity class that refers to a name of an organization that published the CTI.
- *Author*: An entity class that refers to the name of the author of the CTI.
- *Country*: An entity class that refers to the originating country. Such as USA, Canada, Israel, etc.
- *Origin-Type*: An entity class that refers to the type of CTI, such as AAR or Blog etc.

- *Creation-Date*: An entity class that refers to the date of the publishing.
- *Provenance-Score*: A value that refers to the amount of knowledge the CKG posses on that entity.

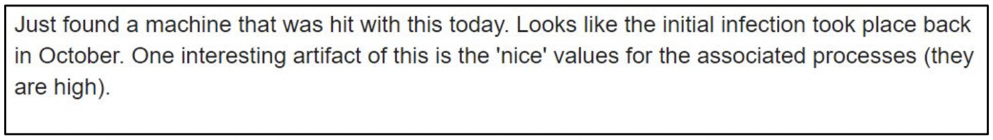
ExP is an updated version of MEE. It was trained using UCO 3.0 to identify the provenance entities from various CTI sources. As UCO 3.0 is developed on top of UCO 2.0, it contains all the entity classes in UCO 2.0 along with the above-listed classes. While executing, the Provenance Extractor parses the CTI sources and classifies the intelligence according to UCO 3.0. While parsing each CTI, we generate a unique *Intelligence-Id* to identify cybersecurity knowledge and allow graph fusion. Based on the Provenance Extractor results and entity vector embeddings obtained using Word2Vec [53], we determine provenance relationships for the particular CTI intelligence. We also keep the count of an entity occurrence to calculate the CKG confidence score for a specific CTI, this informs the value of the *Provenance-Score*.

Chapter 6: Reinforcement Learning Tasks

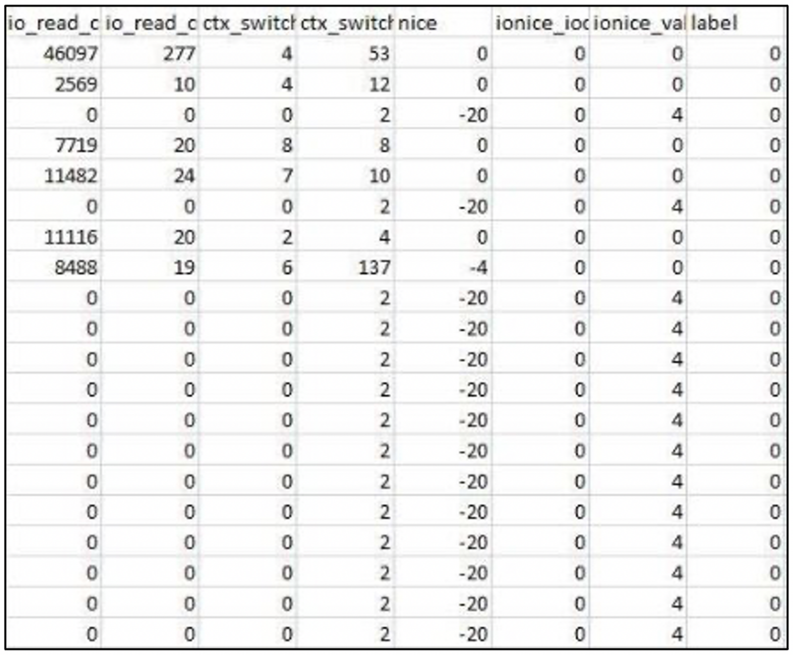
In this chapter, we discuss the methods in which we leverage the information present in the CKGs and the exploratory capabilities of RL for cybersecurity tasks.

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 6.1, 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 6.1, 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.

(a)


Just found a machine that was hit with this today. Looks like the initial infection took place back in October. One interesting artifact of this is the 'nice' values for the associated processes (they are high).

(b)



io_read_c	io_read_c	ctx_switcł	ctx_switcł	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

Figure 6.1: Diagram showing a knowledge source (a) describing a similar malware and the time series data we collected after detonating a malware (b).

6.1 Q-Learning for malware detection with CKG parameters

There are two ways we incorporate prior knowledge into our RL algorithm [74]. The first method are inspired on some of the concepts mentioned by Moreno et al. [75] 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. [75] 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 6.1. We can also manipulate the exploration probabilities with the help of the *T(s)* value.

$$P(s, a_i) = \frac{\exp(Q(s, a_i)/T(s))}{\sum_j \exp(Q(s, a_j)/T(s))} \tag{6.1}$$

In our algorithm, we use the extracted knowledge to tune the probability distribution for exploration of a state-action pair. For example, Equation 6.2 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.

$$P(s, a_i) = 1 - \frac{(nicevalue(a_i) + 20)}{2 \cdot \mid \max_j nicevalue(a_j) \mid} \tag{6.2}$$

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.

Now, we discuss the 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. [76], 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 6.3, 6.4, and 6.5. The first reward function is constructed based on common knowledge and intuition. If the I/O activity or the network activity decreases 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 6.3 and Equation 6.4 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 6.2. This helps the RL algorithm to explore state-action pairs that have high priority nice values leading to a faster convergence.

$$\begin{aligned} Reward(s,a) &\quad += 1 \quad \text{(if I/O write/read decreases} \\ &\quad \text{or KB sent/received decreases)} \end{aligned} \tag{6.3}$$

$$Reward(s,a) \quad -= 1 \quad \text{(otherwise)} \tag{6.4}$$

$$\begin{aligned} Reward(s,a) = {} & w1 \cdot \frac{nicevalue}{20} \\ & + w2 \cdot (numthreads(state, action) \\ & - numthreads(state-1, previousaction)) \end{aligned} \tag{6.5}$$

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 6.5 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 6.3, 6.4, and 6.5 for this experiment. We use a weighted sum of the two prior sources in this experiment.

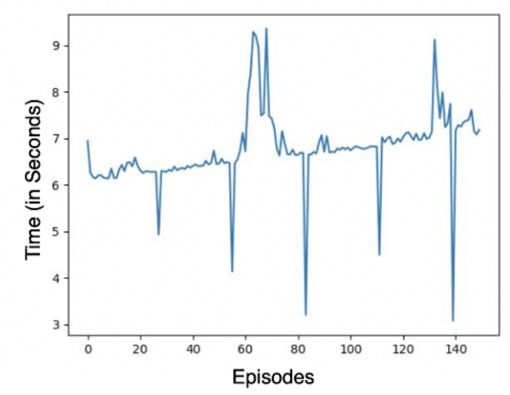
Type	Exp-1	Exp-2	Exp-3	Exp-4
Q-value	-5.1	-5.7	-7	-16.99
Rank	9(99)	11(99)	8(99)	1(99)

Table 6.1: Ranking and Q-values of the known malicious process

In Table 6.1 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.

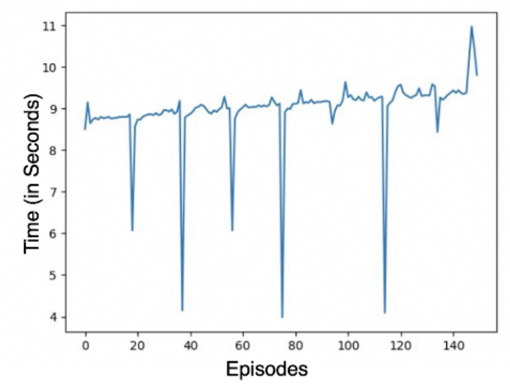
In Figure 6.2, we can see the comparison of time required to complete each episode that consists of 10,000 steps. We argued previously that tuning the exploration probability distribution would lead to a faster convergence. We observe that this is partly true. In sharp dips signify the exploitation phase 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 6.2a than in Figure 6.2b, 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 6.2a, as the RL algorithm's episodes move to the malicious phase. In contrast, the time per episode is more consistent

(a)



For Exp 1 (user-defined reward functions)

(b)



For Exp 2 (tuned exploration probabilities)

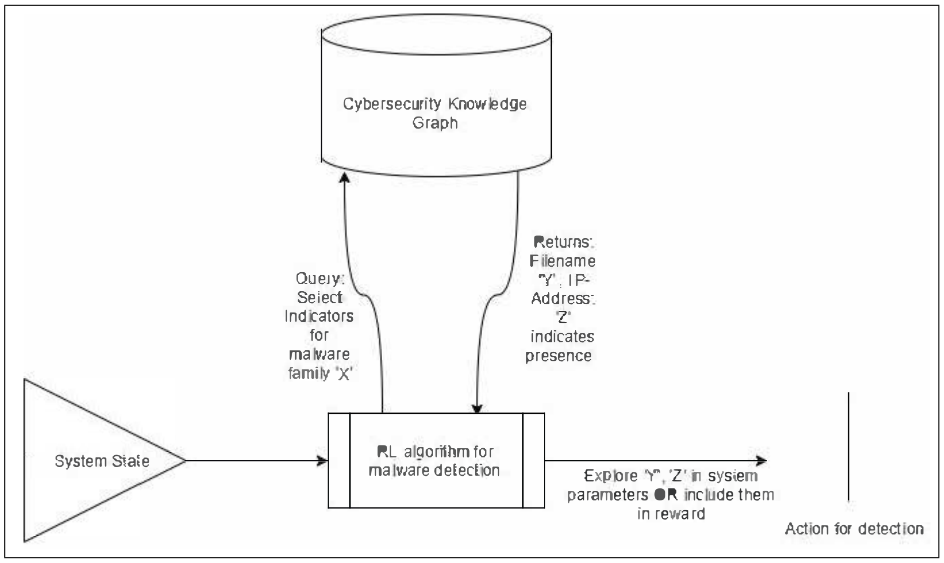
Figure 6.2: Comparison of time required to complete each episode for two experiments

in Figure 6.2b. This is the result of the changed probability distribution that helps the RL algorithm in finding processes faster in the malicious phase.

6.2 Mining RL parameters from CKG

RL algorithms designed for specific cybersecurity tasks need an evaluating criteria. Specifically, reward functions are what drives these algorithms. These algorithms converge to maximize the reward defined to evaluate these tasks. For malware generation, a simple reward function would be to check how successful the generated or mutated malware is to evade an externally trained malware detector. If the malware mutation is unable to evade the malware detector, we observe no reward.

The reward function definitions become more challenging when we are dealing with malware detection and generating malware mitigation strategies. A general framework for detecting malware would be to classify the system parameters as a 'state'. We will define the 'process names' or 'file names' that are getting created as 'actions'. The challenge here would be to quantify the different 'state-action' pairs so that we can use that in the reward function. Some existing methods to evaluate certain attack vectors or vulnerabilities are CVSS (Common Vulnerabilities Scoring System) [77] and DREAD. Although these systems are able to quantify attack-vectors and vulnerabilities, they are very generic. It is difficult for these scoring mechanisms to evaluate 'risks' as opposed to 'severity' of an attack vector. CVSS also places an importance on 'Privileges gained and not attained'. In certain occasions, this is misleading. Severity is also an important factor to be taken into consideration. For example, if a fan falls on someone's head and kills him, the severity of this unfortunate incident is high. However, measuring a 'risk' associated with a ceiling fan over somebody's head would not yield a high score. CKGs will have this information recorded and thus if we were to design a RL algorithm for malware detection, a state-action pair that has 'room' and 'ceiling fan' as parameters, the reward for this pair would be appropriate. CKGs contain context for that attack vector, and also we can leverage the reasoning capabilities of the CKG.



Cybersecurity Knowledge Graph
Query: Select Indicators for malware family 'X'
Returns: Filename 'Y', IP-Address: 'Z' indicates presence
System State
RL algorithm for malware detection
Explore 'Y', 'Z' in system parameters OR include them in reward
Action for detection

Figure 6.3: A general framework for mining RL parameters from CKGs

Figure 6.3 provides a description of a system that incorporates CKGs for RL parameters. An RL algorithm like DQN, when trained, is nothing but a series of matrices that produce the appropriate action as an output. This is too general a framework for the wide variety of malware families that security analysts have to deal with on a daily basis. A general reward function that helps the RL algorithm to update its own weights for producing an appropriate action as an output, may not be sufficient for the variety of malware families. In Figure 6.3, we see that the RL algorithm queries the CKG, with certain arguments in the query specific to the malware family that it is targeting. The query being asked in the figure is about specific indicators for the particular malware family. This is useful information as the RL algorithm can adjust its weights in a way, that these indicators incorporated as parameters in its reward function are used.

An example of such a query is as follows: **SPARQL [78] Query**:

```
SELECT ?x WHERE {
?x a Software; :hasVulnerability ?y .
?z a Malware Family;
?z uses ?y}
```

The SPARQL Query above is basically throwing the question what Softwares have a vulnerability that can be used by a particular malware family. The RL agent can then look for the presence of those softwares in the system and use that information for a specific task.

Another example of a query could be as follows:

```
SELECT ?x WHERE {
  BKG:MalwareFamily M
     BKG:hasParameter ?x.
  ?x BKG:parameterchange
     BKG:increases_maxchange.}
```

The above query asks the CKG (Behavioral Knowledge Graph in this case) that which system parameters creates a change in the max value. The answer to the query is ‘cpu child sys’, ‘io write bytes’. This type of query provides valuable information for the RL algorithm for a cybersecurity task like detection. We can incorporate these parameters in the reward function, for detecting malware belonging to a family M.

6.3 Knowledge Guided Adversarial vs Defensive Policies

Recent advancements in RL have shown promising results for generating adversaries as well as mitigating them with defense mechanisms. RL algorithms can work particularly well for cybersecurity [12] 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’ [79]. 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 [80], 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.

We use the schema of the CKGs that have been used to represent STIX data [20, 81, 82]. These CKGs are populated with information extracted from STIX [33] and contain semantic information about cyberattacks collected from multiple sources, such as TAXII servers as well as from the CVE [83], and CWE [84] datasets. Apart from the knowledge collected from these semi-structured sources, we use a deep learning-based CKG construction pipeline [81] 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 [33], 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.*

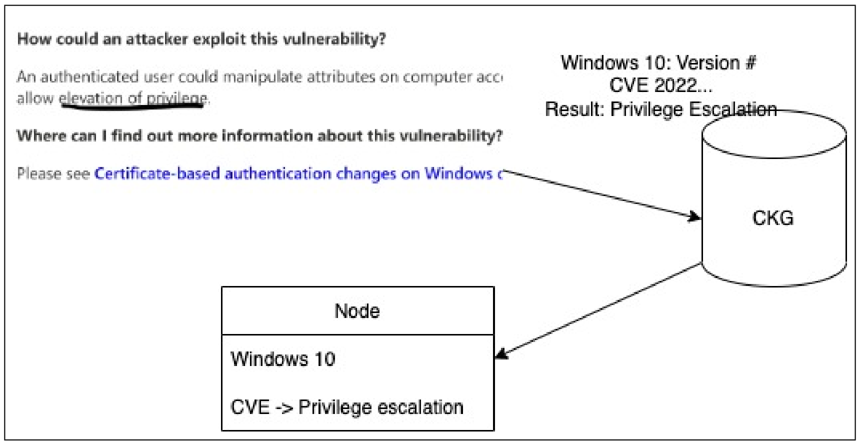

How could an attacker exploit this vulnerability?
An authenticated user could manipulate attributes on computer acc
allow elevation of privilege.
Where can I find out more information about this vulnerability?
Please see Certificate-based authentication changes on Windows c
Windows 10: Version #
CVE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

Figure 6.4: Feeds are parsed to identify vulnerabilities and to generate the network graph for the simulation.

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) [77] 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 6.4, 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.

6.3.1 Two-player RL Agents

The vanilla Deep Q-Network (DQN) [85] 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.

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

In the equation above, Q values for state-action pairs are updated with the

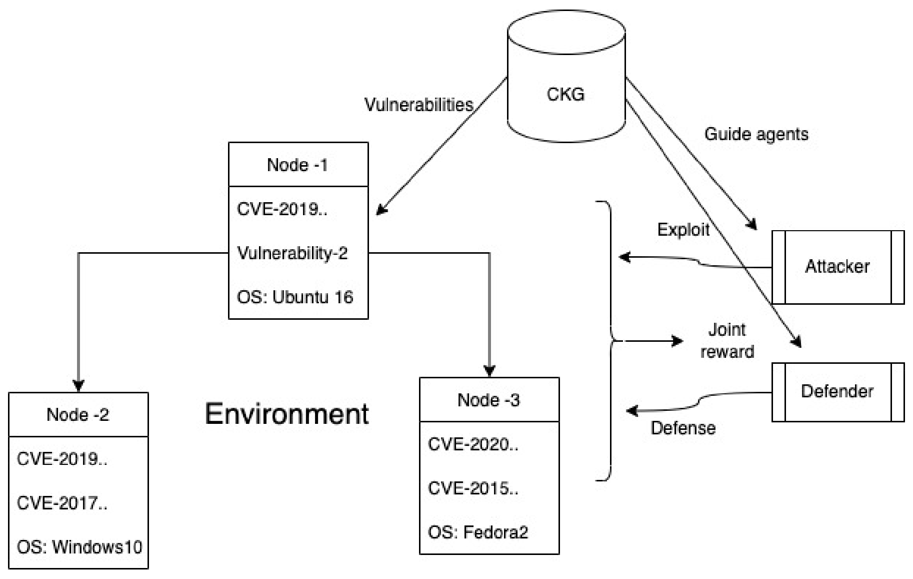


Figure 6.5: An architecture diagram specifying the different components of our model. CKG has information about vulnerabilities of different nodes based on their operating systems and versions. This information is used to populate the node properties. This collection of nodes and properties forms the 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.

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 [86].

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.

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.

$$
\begin{aligned}
&Q(s, a_{attack}, b_{defend}) \leftarrow \\
&\qquad (1-\alpha) \cdot Q(s, a_{attack}, b_{defend}) + \alpha \cdot \{r(s, a_{attack}, b_{defend}) \\
&\qquad + \gamma \cdot \max_{\pi' \in \mathcal{P}(\mathcal{A})} \min_{\phi' \in \mathcal{P}(\mathcal{B})} E_{a'_{attack} \sim \pi', b'_{defend} \sim \phi'} \\
&\qquad [Q\left(s', a'_{attack}, b'_{defend}\right) \qquad (6.6)
\end{aligned}
$$

In Equation 6.6, 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'.

$$
\begin{aligned}
&Tar_i \leftarrow \{r(s, a_{attack}, b_{defend}) \\
&\qquad + \gamma \cdot \max_{\pi' \in \mathcal{P}(\mathcal{A})} \min_{\phi' \in \mathcal{P}(\mathcal{B})} E_{a'_{attack} \sim \pi', b'_{defend} \sim \phi'} \\
&\qquad [Q\left(s', a'_{attack}, b'_{defend}\right) \qquad (6.7)
\end{aligned}
$$

The Minimax DQN has two neural networks: a policy network Q and a target network Tar. The target value is represented by Equation 6.7, whereas the policy network update is shown in Equation 6.8.

$$
\widetilde{Q}_{k+1} \leftarrow \min \sum_{i=1}^{n} \left[Tar_i - Q\left(S_i, a_{attack}, b_{defend}\right)\right]^2 . \qquad (6.8)
$$

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.

6.3.2 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 6.3.1. 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.

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.

Algorithm 2 Knowledge guided minimax DQN

```
1: $C \leftarrow KG(Vuln = X, Node = N)$
2: **if** $C \neq \emptyset$ **then**
3:     $b* = \min_b D[b - C]$
4:     Generate($P_{exp}(b)$)
5: **else**
6:     Generate uniform distribution
7: **end if**
8: Initialize($Q$, $S$)
9: **while** $done \neq True$ **do**
10:     **if** $x < \epsilon$ **then**
11:         $b* \leftarrow Sample(P_{exp}(b))$
12:         $Shuffle(b) \forall b \neq b*$
13:         $b \leftarrow b*$
14:         $Dilate(P_{exp})$
15:         $a \leftarrow Explore(a)$
16:     **else**
17:         $b = \min_b \max_a (Q(s, a, b))$
18:         $a = \max_a \min_b (Q(s, a, b))$
19:     **end if**
20:     $S', R \leftarrow (S, a, b)$
21:     $Tar_i = R + \gamma * \max_{a'} \min_{b'} [Q(S', a', b')]$
22:     $loss = [Tar_i - Q(S, a, b)]^2$
23:     $Q* \leftarrow \min_Q (loss)$
24: **end while**
25: **return** $Q*$
```

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 6.9.

$$P_{exp}(b) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{b - b*}{\sigma}\right)^2\right) \tag{6.9}$$

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 2.

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 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 6.6 shows that our model has better attack rewards than the Random agent.

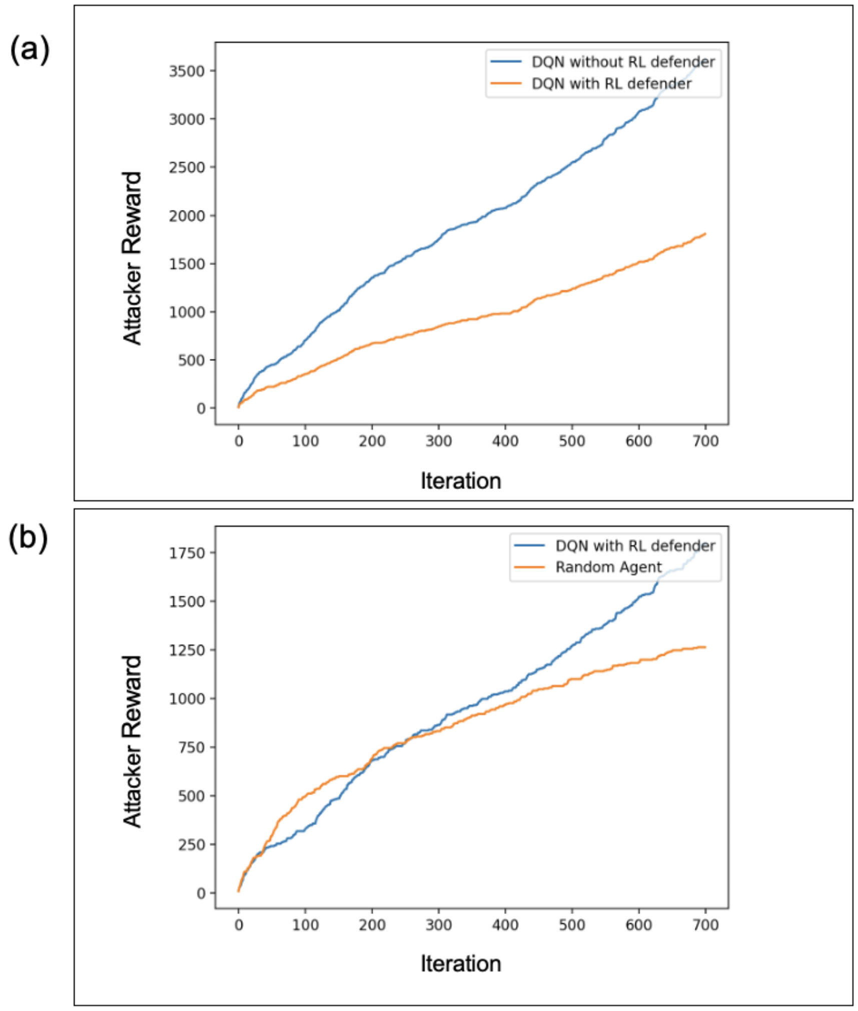


Figure 6.6: Comparison of the cumulative attacker rewards for two scenarios. On the left we see the cumulative **attacker** rewards vs iterations for minimax DQN (DQN with RL defender) compared with a DQN with no defender. To the right we see the cumulative **attacker** rewards of a minimax DQN compared with a Random Agent

One criticism of RL algorithms is that in most episodes, the agents do not observe anything concrete leading to uneventful explorations. In Table 6.2, we see that the episodes with CKG guidance are more eventful. This is because the CKG

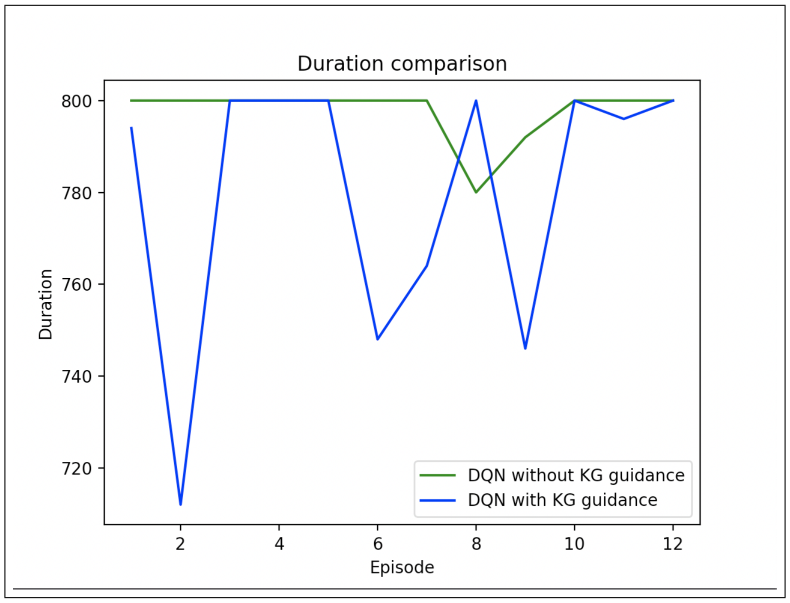


Figure 6.7: Diagram showing the number of steps required to reach a goal at each episode with and without knowledge guidance

is able to inform the agents what needs to be done beforehand. 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 6.7, 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. We also see the effect of knowledge guidance on other RL algorithms in similar simulation environments in Table 6.3.

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

Episodes	DQN (without CKG) % non-zero rewards	DQN (with CKG) % non-zero rewards
0-2	5.1	11.3
3-5	8.4	17.2
6-8	23.3	22.5
9-11	38.3	41.5
11+	37.8	41.7

Table 6.2: Percentage of non-zero rewards with respect to episodes with and without knowledge guidance

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.

6.4 Offline RL and Knowledge guidance

RL can prove to be useful for constantly changing domains like cybersecurity, since it internally creates a simulation map of different variants that may spin off from the data presented for training. This helps deal with the problem of new attacks

	With KG guidance	Without KG guidance
DQN	753+21	794+2
TRPO	788+6	793+3
PPO	784+11	793+5
VPG	767+13	789+2
A3C	745+22	769+4

Table 6.3: Number of iterations per episode for different RL algorithms with and without knowledge guidance

used by adversaries. However, a significant critique of online RL models is that they need to observe a vast number of state-action pairs to generate a generalizable policy, making it impractical for many domains [87], including cybersecurity. However, offline RL techniques [88] have claimed to be just as good with a limited and fixed number of observations of state-action pairs [89]. One of the contributions in this paper is we observe how useful offline RL is for cyber-threat detection. Although offline RL models may produce a policy for cyber-threat detection, it is still limited by partially observed possibilities of cyber threats. For example, if we look at data that describes the behavior of a particular malware in a specific environment, the behavior is limited by the particulars of that environment. The processes created by the malware or the values of system parameters may be different in a different environment or at a different time. It is impossible to simulate the possibilities of malware behavior in all environments. This is where explicit external knowledge

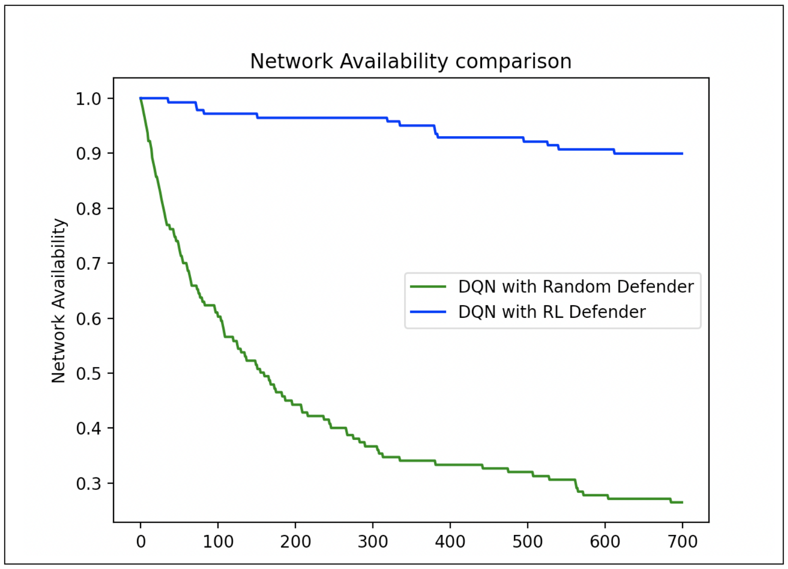


Figure 6.8: Diagram showing the percentage of network available after defender agents perform actions

comes into play. The external knowledge about a domain that an expert has can provide the RL algorithms with key information during the learning process. Our other contribution in this paper is to show how to create knowledge-guided Offline RL algorithms that can leverage the existing body of work that captures explicit domain knowledge about cybersecurity as knowledge graphs [81, 82, 90].

Our research approach uses RL to detect malware. We use a malware behavior dataset containing information on how different system parameters change values when a malware instance from a known malware family is detonated. We also incorporate MOTIF [91], a curated dataset of malware samples and their family names. Our RL algorithm creates a policy to detect a specific malware in a system for which we have data. We can then use an external knowledge source that provides information about other malware in the same family, or possibly a similar malware

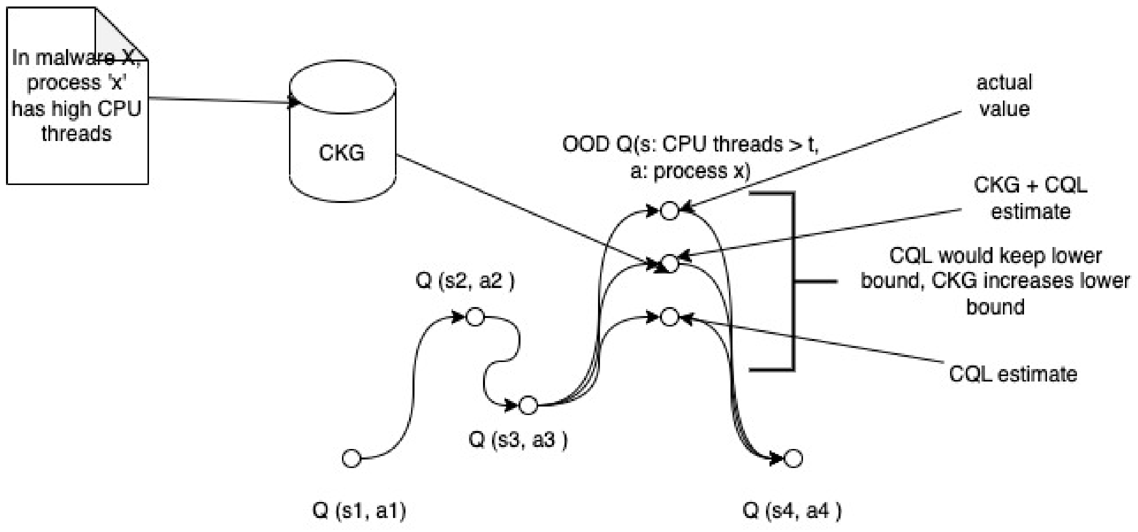

In malware X, process 'x' has high CPU threads
CKG
OOD Q(s: CPU threads > t, a: process x)
actual value
CKG + CQL estimate
CQL would keep lower bound, CKG increases lower bound
CQL estimate
Q (s2, a2)
Q (s3, a3)
Q (s1, a1)
Q (s4, a4)

Figure 6.9: Figure describing the overview of our approach. The input to the CKG is a piece of intelligence describing some characteristics of a malware 'X'. These characteristics are translated to state-action pairs. If these state-action pairs correspond to out-of-distribution (OOD) state-action pairs for the CQL algorithm, we can use it to re-estimate the Q-value for the OOD state-action pair.

family. The RL algorithm can then use this knowledge to make changes to the policy.

CKGs contain information about recent developments in cyber threats. In our prior work, we demonstrated how to generate CKGs from textual threat intelligence and other sources, and used them in downstream tasks [92]. This is also a part of ongoing research that focuses on enriching these knowledge graphs with additional information from Wikidata [93, 94]. In our problem, we use offline RL because observing how system parameters change in real-time is cumbersome and impractical. Offline RL algorithms typically have the problem of overestimating some unseen state-action pairs. To address this, researchers have come up with approaches such as IQL [95] and CQL [87]. CQL downgrades updates on unseen state-action pairs. However, CQL has been argued to be too conservative in updating unseen data. In our domain, we need to update unseen data because the unseen variants of malware data will be out-of-distribution (OOD).

For example, the system parameter observations are made at specific time intervals and not all changes are caught when a malware is active in the system. When a process, or action, is created not all state change observations are available to us. As seen in Figure 6.9, a large number of CPU threads may be created when a particular process is active. This observation, if available to us, can be used for modeling in both online and offline RL algorithms. However, if it is not available to us, offline RL algorithms such as CQL will relatively downgrade this OOD state-action pair. If we have an external knowledge source that creates its own state-action pair distribution K, this can be used to relatively upgrade the OOD state-action pair (high number of CPU threads and process 'x').

Thus, we can create less conservative updates on unseen data if we have evidence of it in the CKG's. Thus we create a β term in the equation if the state-action evidence can be retrieved from the CKG. In the following equation, D is the data distribution, and K is the distribution from the knowledge source.

$$\hat{Q}^{\pi} = \arg\min_{Q} \max_{\mu} \alpha\beta E_{\mathbf{s}\sim D,\mathbf{a}\sim\mu(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s},\mathbf{a})] - \alpha E_{(\mathbf{s},\mathbf{a})\sim D}[Q(\mathbf{s},\mathbf{a})] - \beta E_{(\mathbf{s},\mathbf{a})\sim K}[Q(\mathbf{s},\mathbf{a})]$$

$$+E_{(\mathbf{s},\mathbf{a},\mathbf{s}')\sim D}\left[\left(Q(\mathbf{s},\mathbf{a}) - \left(r(\mathbf{s},\mathbf{a}) + E_{\pi}\left[Q\left(\mathbf{s}',\mathbf{a}'\right)\right]\right)\right)^{2}\right]$$

The equation above minimizes the Bellman error values for all state-action pairs. α is the hyperparameter that tackles the increase of the Q-value estimates of data that is not OOD. β is another hyperparameter that increases the Q-value estimates of the OOD data if the CKG's information supports them.

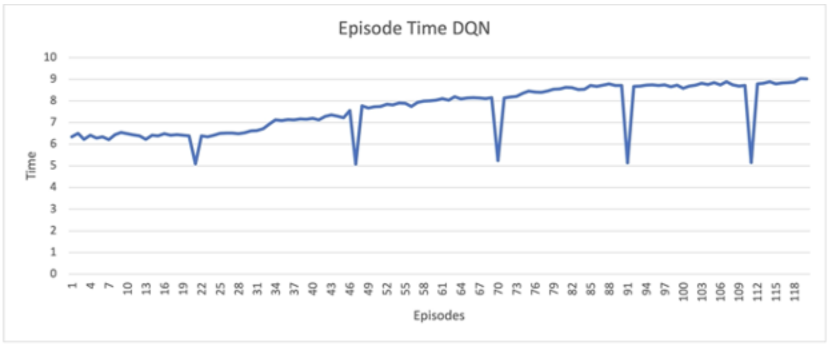
We use malware behavior data that is achieved after detonating a malware sample in a controlled environment. The state space is defined by features of the systems' parameters. In our experiments, we calculate the temporal difference between system parameters and normalize them with min-max normalization. The actions are individual processes that cause these state changes. We propose to find out malicious processes with the help of RL. We have labeled malicious process names for malware samples from different families, and we are going to use information from our CKG for these malware families to help RL algorithms. For example, in Figure 6.9, we see an OSINT source with some information about CPU threads of a process 'x' for a specific malware or a malware belonging to a particular family. As the RL algorithm goes over the state-action pairs in our system parameter data while constructing Q-tables, it may come across a state-action pair that was not encountered in the actual system parameter data. This may be possible due to two reasons. The first reason is the malware sample in this family may not be adhering to the information provided by the CKG about the same family. Secondly, because of the periodic nature of data collection of the system parameter data, this particular state-action pair may not be present in the collected data. This is where the information from CKG comes in, as it lifts the Q-values for those state-action pairs.

6.4.1 Findings

We experimented with 25 malware samples belonging to four malware families. Our first experiment used RL algorithms that work on all types of samples and all malware families, and our second applied family-specific RL algorithms. Each malware sample has time steps (or a number of states) that range from 2163 to 3750. In some cases, we trim the number of states to 3750 due to a high number of redundant states present in the data collected. The action space for the experiment with all malware families is 493, and for individual malware families ranged from 102 to 193. We used user-defined reward functions for these exercises formed by our own understanding of the malware samples.

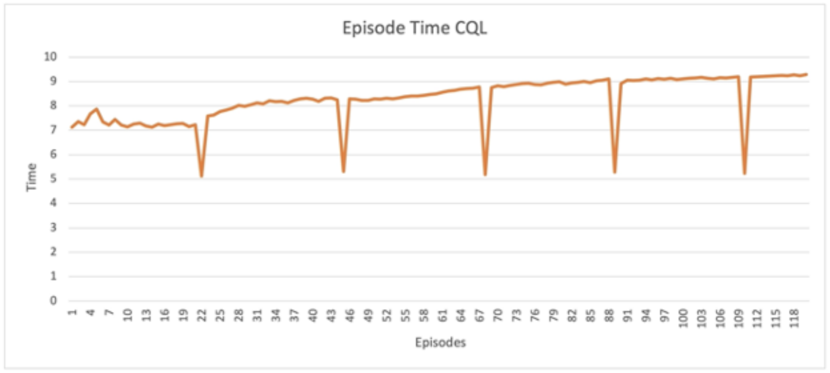
In order to check how CKG influences the performance of CQL, we take two approaches. We plot the cumulative rewards of CQL and CQL with CKG guidance as shown in Figure 6.11. However, it is difficult to quantifiably argue that the difference in cumulative rewards actually results in better detection capabilities since the reward functions are hand-crafted based on our own knowledge of the malware. One algorithm having a higher cumulative reward may be due to the fact that the reward functions are crafted more favorably. One way to check that the Q-values are representative of the actual quality of the state-action pairs is to see how the

(a)



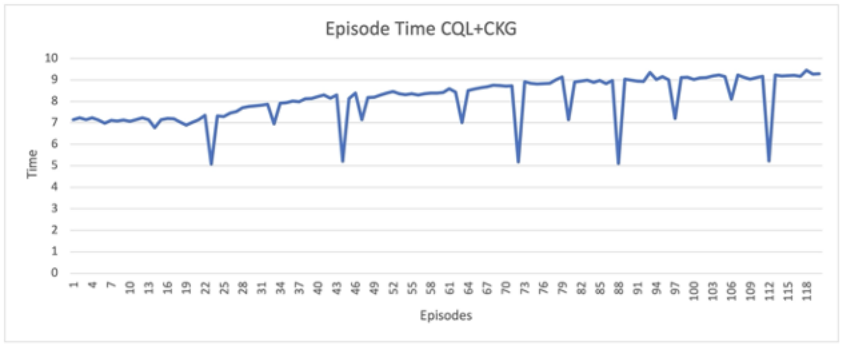
Episode Time for DQN with Q-learning as baseline

(b)



Episode Time for CQL

(c)



Episode Time for CQL with knowledge guidance

Figure 6.10: Comparison of time required to complete each episode for different experiments

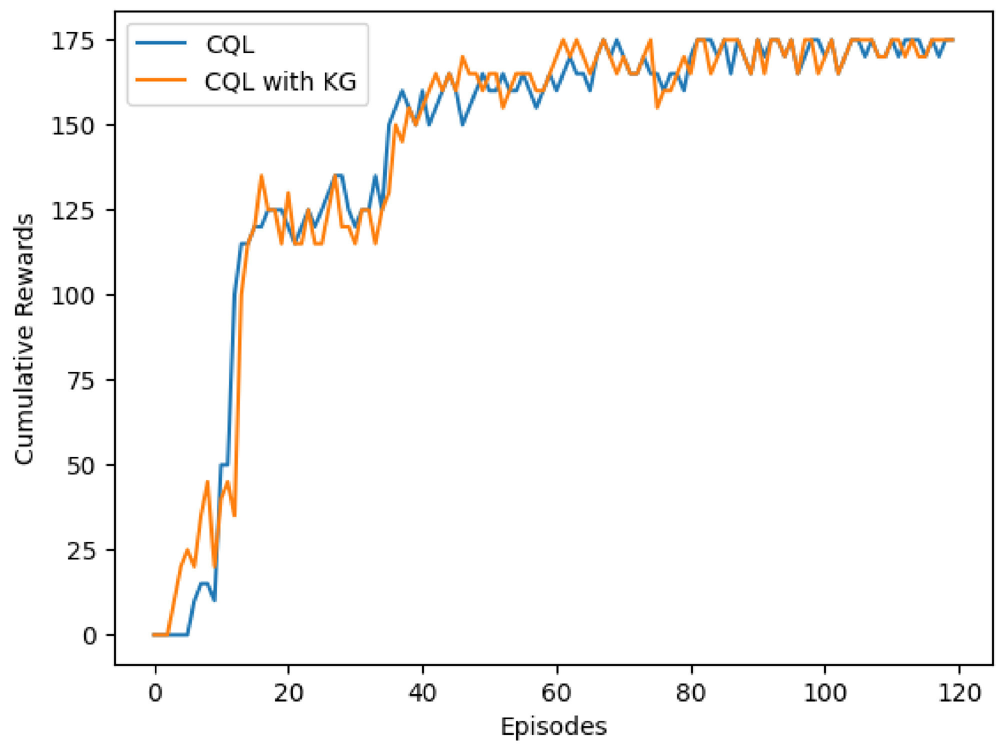


Figure 6.11: Comparison of cumulative rewards between CQL and CQL with KG guidance

associated Q-values are for state-action pairs involving malicious processes. As we mentioned before, we labeled some of the malicious processes in the data collected. We check the top 10% of the Q-values for all state-action pairs in our dataset and observe what percentage of them involve malicious processes.

Ideally, the malicious processes should have the highest Q-values because our RL training is designed in such a way. In Table 6.4, we observe that the Q-values for RL models are trained for all families and RL models trained for individual families. The CQL model trained for all models together does not benefit from CKG

Malware Family	CQL	CQL+CKG (β=0.05)	CQL+CKG (β=0.1)
All	0.74	0.72	0.72
Acidbox	0.68	0.73	0.71
RogueRobin	0.87	0.89	0.91
vhd	0.93	0.91	0.91
hiddenwasp	0.86	0.93	0.93

Table 6.4: Hits@10 calculated for Q-values of malicious processes

Malware Family	CQL	CQL+CKG (beta = 0.05)	CQL+CKG (beta = 1)	Offline DQN
All malware	0.73	0.75	0.73	0.73
Acidbox	0.59	0.62	0.63	0.60
RogueRobin	0.84	0.88	0.86	0.79
Vhd	0.90	0.86	0.89	0.87
Hiddenwasp	0.79	0.85	0.86	0.72

Table 6.5: Detection rates of offline RL algorithms for malware families with and without knowledge guidance

information. This may be because of the generic nature of the CKG's information for all models. A particular process name with a higher Q-value may be relevant for a particular family, but it may mistakenly raise the Q-values for a state-action pair for a different family. However, if we look at specific malware families, we see three of the four malware families observe some lift in scores. We also note how the variable β affects the scores. This variable controls by what factor we raise the scores of the Q-values for OOD state-action pairs that are suggested by the CKG. Raising the value of β may sometimes decrease the score because, in some cases, it may artificially inflate the values of some state-action pairs.

We also observe the time required to complete each episode. In Figure 6.10, we see the episode time for DQN as baseline, CQL, and CQL+CKG with $\beta = 0.5$. The periodic dips that we see are due to exploration-exploitation. As we see, the average time to complete each episode is higher for CQL than for DQN. This is mostly due to higher computation costs for each iteration. When we compare CQL with CQL with CKG, we observe that there are more dips in the episode times. We stop each episode when the cumulative rewards reach a threshold, indicating that the malicious process has been found or when all the states are covered. In some cases, CKG might have helped to raise the cumulative rewards and made it reach the threshold faster than vanilla CQL. The average episode time, even though the computation is more complex in CQL+CKG, is lower because of these dips. However, in some cases, CKG also contributes to the distributional shift problem of RL training leading to higher episode times.

Our results show that for some malware families, such as *hiddenwasp*, we observe a lift of 7% in detection capabilities. In Table 6.5, we see the detection rates of different offline RL algorithms for specific malware families. For acidbox, hiddenwasp, and roguerobin families we observe an average improvement of about 4% with the help of knowledge guidance. The ability of KG guidance in increasing the quality of Q-learning depends on how representative the knowledge is and how easy it is to transfer the knowledge to RL parameters. Variance between samples of malware families also play an important role. Some of the problems that we faced in this can be mitigated by using a wider range of knowledge. We are currently enriching our cyber knowledge bases with the help of Wikidata [93, 94]. It will be interesting to see how more knowledge about malware families affects the generalizability of offline RL models trained with knowledge guidance. In this paper, we talked about specific cybersecurity tasks, such as malicious process detection. We would like to extend this to other cybersecurity tasks like cyber attacks and defenses.

Chapter 7: Conclusion and Future Work

The dissertation research focuses on two main components: utilizing Natural Language Processing (NLP) techniques to extract structured knowledge from unstructured text, and using this knowledge to guide data-driven machine learning (ML) and Reinforcement Learning (RL) algorithms in the field of cybersecurity.

In the first part, the research involves gathering Open-Source Intelligence (OSINT) using NLP techniques. Unstructured text, such as social media posts and After Action Reports, is processed to extract semantic triples that represent relationships between entities. This extracted knowledge is used to construct a knowledge graph, which provides a structured representation of the domain-specific information.

To ensure the accuracy and reliability of the knowledge graph, additional steps are taken. Trusted information is fused with the extracted knowledge, score metrics and provenance related information are included to filter out potentially poisoned or incorrect data. Graph neural network models are developed to detect and mitigate data poisoning, enhancing the overall quality and reliability of the knowledge graph.

In the second part of the research, the knowledge graph is utilized to guide data-driven ML and RL algorithms. The goal is to leverage the tacit and implicit knowledge present in the graph to enhance the performance and efficiency of these algorithms. The knowledge graph serves as a source of prior knowledge that can guide the exploration and decision-making processes of the algorithms.

In the case of RL algorithms, the knowledge graph is used to influence the exploration phase, leading to the discovery of new information about cyber-attacks. Incorporating prior knowledge in the reward function of RL algorithms and attesting states favored by the knowledge sources result in faster convergence, 8% and improved conclusions. The research demonstrates that including prior knowledge in RL algorithms can decrease mean episode time.

Additionally, hybrid AI algorithms are developed that combine semantic knowledge with data-driven algorithms. Offline RL algorithms are utilized, and the impact of prior knowledge on their efficiency is analyzed. The experiments conducted on large cybersecurity datasets show that incorporating prior knowledge leads to increased efficiency in RL algorithms, improving their performance by approximately 4%. We study the effectiveness of RL in detecting unseen malware samples, as well as their effectiveness in generating mitigation steps in simulated environments. We also observe that the mitigation steps are more precise when guided by knowledge.

Overall, the dissertation research showcases the potential of NLP techniques in extracting structured cyber knowledge from unstructured text and demonstrates how this knowledge can guide data-driven ML and RL algorithms in the field of cybersecurity. By leveraging the information contained in knowledge graphs, these algorithms can make more informed decisions, enhance their performance, and expedite the training of the models.

This research can be expanded to domain-independent neuro-symbolic approaches. The current interest lies in the identification of prompts or keywords that can conditionally produce desired outputs from generative models. I propose achieving this through knowledge transfer from prior knowledge sources. Additionally, the emergence of Language Model-based Models (LLMs) has introduced challenges with incorrect language generation. By effectively maintaining the correctness of knowledge graph assertions, these issues can be addressed using knowledge sources.

The second area of research that can be expanded from this is transfer learning from knowledge graphs to other machine learning (ML) models for cybersecurity. With the increasing prevalence of zero-day vulnerabilities, constructing models that maintain effectiveness in the future using existing data proves difficult. Often, these models necessitate retraining with new data to sustain performance levels. For instance, an attack-detection model trained on a specific dataset may lose effectiveness when a new attack pattern emerges. The conventional approach of gathering new data and retraining the model is costly. However, if we possess new information about novel attack patterns, whether in unstructured (e.g., text) or structured (e.g., knowledge graphs) formats, we can directly update the model through a transfer learning mechanism. This research has substantial potential in addressing significant data science challenges, and considerable work remains to be done in this domain.

The third research area that can be worked on is online/continual learning. Given the increasing ubiquity of ML models, the constant need for updates poses feasibility challenges. In some cases, online learning becomes necessary as the performance of an ML model trained on historical data significantly deteriorates before new data becomes available. In such situations, online learning represents the only viable option. However, if we can mitigate performance degradation before new data arrives, it would provide a substantial advantage. It is intriguing to investigate whether external knowledge sources can anticipate certain characteristics of new data and effectively adjust ML model parameters. For example, in the context of an ML model detecting cyber-attacks, the current model may assign low importance to the country of origin. However, if we obtain information indicating a new attack variant specific to a particular country, we can proactively tune the model accordingly prior to acquiring the dataset containing the new cyber-attack variant.

In summary, my research aspirations involve domain-independent neuro-symbolic approaches, transfer learning from knowledge graphs to enhance cybersecurity models, and advancements in online/continual learning techniques. Each of these areas offers unique opportunities for addressing crucial challenges in AI and cybersecurity, contributing to the growing body of knowledge in the field.

Bibliography

[1] Security Boulevard. Cybercrime to cost over $ 10 trillion by 2025. `https://securityboulevard.com/2021/03/cybercrime-to-cost-over-10-trillion-by-2025/`, 2021.

[2] Embroker. 2021 must-know cyber attack statistics and trends. `https://www.embroker.com/blog/cyber-attack-statistics/`, 2021.

[3] NPR. A 'worst nightmare' cyberattack: The untold story of the solarwinds hack. `https://www.npr.org/2021/04/16/985439655/a-worst-nightmare-cyberattack-the-untold-story-of-the-solarwinds-hack`, 2021.

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

[5] Palo Alto Networks. The growing role of machine learning in cybersecurity. `https://www.securityroundtable.org/the-growing-role-of-machine-learning-in-cybersecurity/`, 2023.

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

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

[8] 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.

[9] SECUDE. Is data collaboration the key to improving cybersecurity? `https://secude.com/is-collaboration-key-to-improving-cybersecurity/`.

[10] MIT Press. The colonial pipeline ransomware hackers had a secret weapon: self-promoting cybersecurity firms. `https://www.technologyreview.com/2021/05/24/1025195/colonial-pipeline-ransomware-bitdefender/`, May 2021.

[11] Gaurav Sood. *virustotal: R Client for the virustotal API*, 2021. R package version 0.2.2.

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

[13] Microsoft. How artificial intelligence stopped an emotet outbreak. `https://www.microsoft.com/security/blog/2018/02/14/how-artificial-intelligence-stopped-an-emotet-outbreak/`, February 2018.

[14] Subbarao Kambhampati. Polanyi's revenge and ai's new romance with tacit knowledge. *Commun. ACM*, 64(2):31–32, January 2021.

[15] MIT. Ai pioneer geoff hinton: "deep learning is going to be able to do everything". `AIpioneerGeoffHinton:œDeeplearningisgoingtobeabletodoeverything`, November 2020.

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

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

[18] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[19] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the Twenty-First International Conference on Machine Learning*, ICML '04, page 1, New York, NY, USA, 2004. Association for Computing Machinery.

[20] 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.

[21] 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.

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

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

[24] 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.

[25] 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.

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

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] Zareen Syed, Ankur Padia, Tim Finin, Lisa Mathews, and Anupam Joshi. Uco: A unified cybersecurity ontology. In *AAAI Workshop on Artificial Intelligence for Cyber Security*. AAAI Press, 2016.

[32] Oasis group. Stix 1.0 documentation. url-https://stixproject.github.io/documentation/, May 2018.

[33] Oasis group. Stix 2.0 documentation. `https://oasis-open.github.io/cti-documentation/stix/examples.html`, May 2013.

[34] MITRE. CVELIST project. `https://github.com/CVEProject/cvelist`, May 2020.

[35] NisT. Nvd. `https://nvd.nist.gov/`.

[36] MITRE. Cwe mitre. `https://cwe.mitre.org/data/index.html`, May 2018.

[37] Kaspersky. Kaspersky reports. `https://usa.kaspersky.com/enterprise-security/apt-intelligence-reporting`, May 2020.

[38] FireEye. Fireeye reports. `https://usa.kaspersky.com/enterprise-security/apt-intelligence-reporting`, May 2020.

[39] Microsoft. Microsoft security bulletin. `https://msrc-blog.microsoft.com/tag/security-bulletin/`.

[40] Adobe. Adobe security bulletin. `https://helpx.adobe.com/security.html`.

[41] Aritran Piplai, Sudip Mittal, Mahmoud Abdelsalam, Maanak Gupta, Anupam Joshi, and Tim Finin. Knowledge enrichment by fusing representations for malware threat intelligence and behavior. In *International Conference on Intelligence and Security Informatics (ISI)*, pages 1–6. IEEE, 2020.

[42] Nitika Khurana, Sudip Mittal, Aritran Piplai, and Anupam Joshi. Preventing poisoning attacks on AI based threat intelligence systems. In *Int. Workshop on Machine Learning for Signal Processing*. IEEE, 2019.

[43] Asif Ekbal and Sivaji Bandyopadhyay. Named entity recognition using support vector machine: A language independent approach. *World Academy of Science, Engineering and Technology*, 39, 2010.

[44] Jason Chiu and Eric Nichols. Named entity recognition with bidirectional lstm-cnns. *Transactions of the Association for Computational Linguistics*, 4:357–370, 2016.

[45] Houssem Gasmi, Abdelaziz Bouras, and Jannik Laval. Lstm recurrent neural networks for cybersecurity named entity recognition. *13th International Conference on Software Engineering Advances*, 2018.

[46] Robert Bridges, Corinne Jones, Michael Iannacone, and John Goodall. Automatic labeling for entity extraction in cyber security. *arXiv:1308.4941 [cs.IR]*, 2013.

[47] Chunping Ma, Huafei Zheng, Pengjun Xie, Chen Li, Linlin Li, and Si Luo. Neural sequence labeling with linguistic features. In *12th International Workshop on Semantic Evaluation*, page 707–711, 2018.

[48] Manikandan R, Krishna Madgula, and Snehanshu Saha. Cybersecurity text analysis using convolutional neural network and conditional random fields. In *12th International Workshop on Semantic Evaluation*, pages 868–873. ACL, 2018.

[49] George Hsieh Satyanarayan Vadapalli and Kevin Nauer. Twitterosint:automated cybersecurity threat intelligence collection and analysis using twitter data. *International Conference on Security and Management*, 2018.

[50] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *28th AAAI Conference on Artificial Intelligence*, 2014.

[51] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Durán, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *26th International Conference on Neural Information Processing Systems, Vol. 2*, pages 2787–2795, 2013.

[52] Guoliang Ji, Kang Liu, Shizhu He, and Jun Zhao. Knowledge graph completion with adaptive sparse transfer matrix. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[53] 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.

[54] Xuxian Jiang, Xinyuan Wang, and Dongyan Xu. Stealthy malware detection through vmm-based "out-of-the-box" semantic view reconstruction. In *14th ACM conference on Computer and communications security*. ACM, 2007.

[55] 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.

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

[57] 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.

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

[59] 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.

[60] 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.

[61] 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.

[62] 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.

[63] 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.

[64] 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.

[65] 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.

[66] 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.

[67] Stanisław Saganowski. Cybersecurity NER corpus 2019. 2020.

[68] Anantaa Kotal Soham Dasgupta, Aritran Piplai and Anupam Joshi. A Comparative Study of Deep Learning based Named Entity Recognition Algorithms for Cybersecurity. In *IEEE International Conference on Big Data 2020*. IEEE, December 2020.

[69] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F Patel-Schneider, and Sebastian Rudolph. OWL 2 web ontology language primer, 2012.

[70] *Generating Fake Cyber Threat Intelligence Using Transformer-Based Models*. IEEE, July 2021.

[71] Priyanka Ranade Soham Dasgupta, Aritran Piplai and Anupam Joshi. Cybersecurity Knowledge Graph Improvement with Graph Neural Networks. In *International Conference on Big Data*. IEEE, December 2021.

[72] Sudip Mittal Shaswata Mitra, Aritran Piplai and Anupam Joshi. Combating Fake Cyber Threat Intelligence using Provenance in Cybersecurity Knowledge Graphs. In *IEEE International Conference on Big Data 2021*. IEEE, January 2022.

[73] M. Lisa Mathews Tim Finin Zareen Syed, Ankur Padia and Anupam Joshi. UCO: A Unified Cybersecurity Ontology. In *Proceedings of the AAAI Workshop on Artificial Intelligence for Cyber Security*. AAAI Press, February 2016.

[74] Aritran Piplai, Priyanka Ranade, Anantaa Kotal, Sudip Mittal, Sandeep Nair Narayanan, and Anupam Joshi. Using Knowledge Graphs and Reinforcement Learning for Malware Analysis. In *IEEE International Conference on Big Data 2020*, December 2020.

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

[76] 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.

[77] NIST. Common vulnerability scoring system. `https://nvd.nist.gov/vuln-metrics/cvss`, 2019.

[78] W3. Sparql query language. `https://www.w3.org/TR/rdf-sparql-query/`, May 2020.

[79] Microsoft. CyberBattleSim. `https://github.com/micro-soft/CyberBattleSim`, April 2021.

[80] Kayode Fasaye Anupam Joshi Tim Finin Aritran Piplai, Mike Anoruo and Ahmad Ridley. Knowledge guided Two-player Reinforcement Learning for Cyber Attacks and Defenses. In *International Conference on Machine Learning and Applications*. IEEE, December 2022.

[81] Aritran Piplai, Sudip Mittal, Anupam Joshi, Tim Finin, James Holt, and Richard Zak. Creating cybersecurity knowledge graphs from malware after action reports. *IEEE Access*, 8:211691–211703, 2020.

[82] Tim Finin Lisa Mathews Zareen Syed, Ankur Padia and Anupam Joshi. Uco: A unified cybersecurity ontology. *The Workshops of the Thirtieth AAAI Conference on Artificial Intelligence Artificial Intelligence for Cyber Security: Technical Report WS-16-03*, 2016.

[83] MITRE. Cvelist project. `https://github.com/CVEProject/cvelist`, May 2013.

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

[85] 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.

[86] 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.

[87] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.

[88] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[89] Ashvin Nair, Abhishek Gupta, Murtaza Dalal, and Sergey Levine. Awac: Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

[90] 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.

[91] Robert J Joyce, Dev Amlani, Charles Nicholas, and Edward Raff. Motif: A large malware reference dataset with ground truth family labels. *arXiv preprint arXiv:2111.15031*, 2021.

[92] Aritran Piplai, Mike Anoruo, Kayode Fasaye, Anupam Joshi, Tim Finin, and Ahmad Ridley. Knowledge guided two-player reinforcement learning for cyber attacks and defenses. In *International Conference on Machine Learning and Applications*, 2022.

[93] Casey Hanks, Michael Maiden, Priyanka Ranade, Tim Finin, and Anupam Joshi. Recognizing and Extracting Cybersecurity Entities from Text. In *Workshop on Machine Learning for Cybersecurity, International Conference on Machine Learning*. PLMR, July 2022.

[94] Varish Mulwad, Tim Finin, Vijay S. Kumar, Jenny Weisenberg Williams, Sharad Dixit, , and Anupam Joshi. A Practical Entity Linking System for Tables in Scientific Literature. In *3rd Workshop on Scientific Document Understanding*. AAAI, February 2023.

[95] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169*, 2021.