Multiple Level Action Embedding for Penetration Testing

Hoang Viet Nguyen
hoang@cysec.cs.ritsumei.ac.jp
Graduate School of Information Science and Engineering, Ritsumeikan University
Japan

Hai Ngoc Nguyen
hai@cysec.cs.ritsumei.ac.jp
Graduate School of Information Science and Engineering, Ritsumeikan University
Japan

Tetsutaro Uehara
uehara@cysec.cs.ritsumei.ac.jp
College of Information Science and Engineering, Ritsumeikan University
Japan

ABSTRACT

Penetration Testing (PT) is one of the most effective and widely used methods to increase the defence of a system by looking for potential vulnerabilities. Reinforcement learning (RL), a powerful type of machine learning in self-decision making, is demonstrated to be applicable in PT to increase automation as well as reduce implementation costs. However, RL algorithms are still having difficulty on PT problems which have large network size and high complexity. This paper proposes a multiple level action embedding applied with Wolpertinger architect (WA) to enhance the accuracy and performance of the RL, especially in large and complicated environments. The main purpose of the action embedding is to be able to represent the elements in the RL action space as an n-dimensional vector while preserving their properties and accurately representing the relationship between them. Experiments are conducted to evaluate the logical accuracy of the action embedding. The deep Q-network algorithm is also used as a baseline for comparing with WA using the multiple level action embedding.

CCS CONCEPTS

• **Theory of computation** → **Reinforcement learning**; **Markov decision processes**; • **Security and privacy** → **Penetration testing**.

KEYWORDS

deep reinforcement learning, deep Q-network, network simulation

ACM Reference Format:
Hoang Viet Nguyen, Hai Ngoc Nguyen, and Tetsutaro Uehara. 2020. Multiple Level Action Embedding for Penetration Testing. In *The 4th International Conference on Future Networks and Distributed Systems (ICFNDS) (ICFNDS '20), November 26–27, 2020, St.Petersburg, Russian Federation.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3440749.3442660

1 INTRODUCTION

Currently, the rapid growth of networking is causing more and more attack threats in cyberspace. Many network security approaches have been used to improve the security level of systems. One of the most effective methods in assessing the defensive capabilities of network security systems is penetration testing (PT). It is a popular approach with the main goal is preventing potential attacks from network systems. By reproducing what actual attacker could do in real-life, PT is able to find all the vulnerabilities existed on the target networks. Thus, the network security team can fix security holes before adversaries can attack the system.

Even though the benefits of PT in secure network systems are undeniable, existed obstacles keep organizations from getting access to its benefits. Firstly, PT is regularly viewed as a costly technique which uses a lot of resources such as budget and manpower. Performing PT also required engineers who have strong experience in network security. Secondly, it is a hard task dealing complex network environments which have a large size and difficult to attack. Even for the experts, finding all of the vulnerabilities currently present on the system is beyond their capabilities.

For this reason, the use of automatic PT methods is necessary to offload the workload that humans have to perform. Additionally, it also provides more accuracy in determining the vulnerabilities than the manual method. Tenable, Nessus and Metasploit are considered as useful tools in supporting PT. These tools, however, only assist security professionals in scanning vulnerabilities rather than helping them exploit systems. In most of the time, experts are the one who has responsibilities on PT process from information gathering to exploitation. In recent years, It is becoming increasingly common to use machine learning to replace human in complex tasks. ML has been found to be able, in some situations, to cope with complicated issues more easily and reliably than humans.

Reinforcement learning (RL) a type of ML has been shown is able to plan and present correct actions in an unstable environment. RL strives to find out viable options for environmental adaptation and derive the maximum value by continuously interacting with the environment and optimizing the action at every turn. Thus, the ability of RL shows that it is completely sufficient to address PT issues. Currently, this direction is pursued by researchers and showing some promising results. Schwartz et al. has proven that the RL algorithm can automatically exploit vulnerabilities on networks and deploy attacks on target machines [14]. On the other hand, Ghanem et al. suggested the capacity to integrate RL with existing PT systems to execute tasks without human intervention [3].

However, all previous approaches show the limitation in the size and complexity of the environment that can be addressed. Among the mentioned articles, the maximum number of machines used for testing the algorithm's performance was 100 machines [3]. Schwartz et al. showed that the figure was lower, ranging from 30-50 machines [14]. Besides, for complex networks such as networks containing services that are difficult to attack (low rate of attack success), the

ICFNDS '20, November 26–27, 2020, St.Petersburg, Russian Federation

ACM ISBN 978-1-4503-8886-3/20/11…$15.00
https://doi.org/10.1145/3440749.3442660

performance of current algorithms is not good as expected. Besides, for networks with too many vulnerabilities, finding all of these vulnerabilities is also not an easy work.

In this paper, we propose a multiple level action embedding that can be used with Wolpertinger architecture (WA) to increase the accuracy in complex environments and the network size that can be solved by RL [2]. The multiple level action embedding represents any action in the action space into an n-dimensional vector by using three levels: action characteristics, network structure and services vulnerabilities. The embedding can accurately represent the action in vector based on its characteristics and find related actions that ensure logically correct requirements. We ran a variety of experiments to evaluate the effectiveness of the action embedding. The results show that even with a complex network with services having a low attack success rate of 20% and a sensitive host rate of up to 100%, the WA using the action embedding maintains its performance above 90%. For a large size environment, its performance is also very good with an accuracy rate of about 70% for the network having 256 hosts.

This paper is organized as follows. Section 2 introduces basic knowledge about RL and review some related works. In section 3, The design of the multiple level action embedding will be presented in detail. Section 4 describes the experimental setup and the results of these experiments. Finally, the conclusion of the paper will be presented in section 5.

2 RELATED WORK

Initially, RL was mainly used in game theory with the aim of finding the best strategy via Markov decision theory and delayed reinforcement [6]. Later, the method of applying a deep neural network to Reinforcement learning, called Deep Q-network (DQN), proved to be extremely effective for practical problems [11]. Since then many scientists have pursued this method, trying to improve the speed as well as solve the remaining problems of RL. Currently, there are several commonly used RL algorithms including double deep Q-network (DDQN), dueling deep Q-network and dueling double deep Q-network (D3QN) [5, 7, 15]. These algorithms focus on solving the overestimation problem of action values and trying to achieve better policies. Some additional techniques such as experience replay and prioritized experience replay are also used to support convergence and smooth out the learning process [8, 13].

Although the RL has undergone many improvements since its introduction, the current RL still faces a scalability challenge. RL approaches are only appropriate for environments with low-dimensions. Variant DQN was introduced by Zhao et al. try to solve this problem by reducing the size of the output layer in the neural network [16]. Dulac-Arnold et al. propose Wolpertinger architecture (WA) handle the same problem using the action embedding and kNN layers [2]. Wolpertinger architecture uses an actor-critic algorithm [10] with the main idea of finding the maximum Q-value from similar actions that increase the chances of the agent finding the best action. WA has been shown to be effective in problems with large discrete action spaces.

In recent years, many researchers have tried to take advantage of the power of RL to solve penetration testing problems. Schwartz et al. propose the network attack simulator and perform many experiments to evaluate the ability of DQN algorithm to solve PT problems [14]. The paper points out that although RL is capable of applying to PT, with large or complex networks, the performance of algorithms is not as high as expected. IAPTS, a Partially observable Markov decision process (POMDP) approach proposed by Ghanem, tried to deal with the same problems [3]. IAPTS is expected to improve the performance of the RL algorithm on large networks. Unfortunately, the experiment was only tested on networks with 10-100 machines. Currently, the ability to solve in a large and complex environment is still a challenge when applying RL to PT.

3 THE PROPOSED EMBEDDING APPROACH

In this section, we focus on two main parts: an overview of the Wolpertinger architecture (WA) and multiple level action embedding design. First of all, the WA, a powerful algorithm that has been proven to be able to handle problems in large action space [2], will be discussed in detail. To work with WA, an action embedding built specifically for a certain problem is necessary. Thus, the paper proposed a multiple level action embedding which can describe the properties of actions within the action space of PT problems. The design of the action embedding will be described in section 3.2

3.1 An overview about Wolpertinger architecture

For all RL problems, the size of the environment is always a matter of scrutiny. It is not only the crucial part that determines the efficiency of the algorithm, but also determines whether the problem can be solved. Many scientists are pursuing this research direction and achieved some success. Wolpertinger architecture (WA) a method of applying deep reinforcement learning and action embedding is seen as one of the most effective methods. While other articles focus on shrinking the state space size of the environment, WA mainly focuses on solving problems with large discrete action spaces.

Before diving into WA, we will discuss a deep reinforcement learning (DRL) algorithm that is considered the core of the architecture called the actor-critic algorithm [10]. The actor-critic algorithm is one of the most popular algorithms which is widely applied in RL. Comparing to the other DRL algorithms such as DQN and Dueling DQN, the actor-critic algorithm uses two different neural networks including a critic network and an actor-network. While the actor-network is in charge of choosing the appropriate action for each step, the critic network takes responsibilities of evaluating how good the action is by calculating the action value and state value of this action. Upon receiving the valuation from the critic network, the actor-network will update its policy to take more appropriate action for the next step. This solution has been shown to be effective when used in unstable environments [11].

Although the actor-critic algorithm has powerful in solving problems, it is still having a hard phrase dealing with complicated environments. Thus, Gabriel et al. proposed WA leveraging the ability of the actor-critic algorithm but improve its weakness when applying to the large action space problems. To increase the ability of the algorithm, WA uses an additional layer called action embedding. The main purpose of this layer is to increase the ability to choose the correct action at each step by considering not only the proto action selected by the actor-network but also the actions associated

with it. In other words, the action embedding layer acts as an extra filter to aid in action selection of the actor-network.

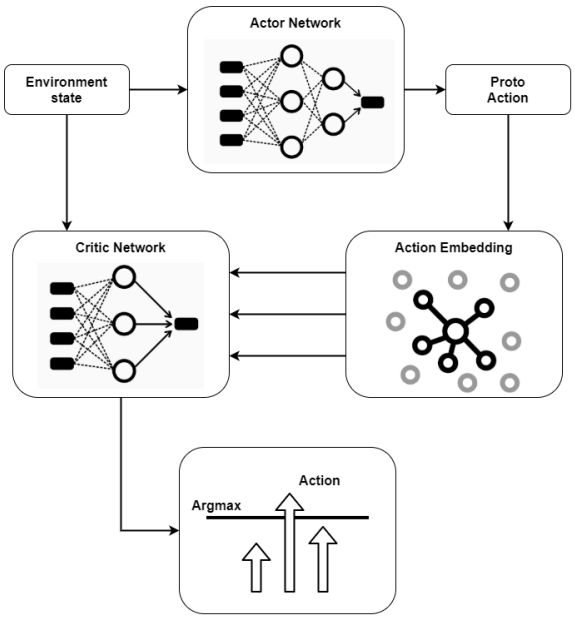


Figure 1: Wolpertinger Architecture

Figure 1 describes how WA works. First of all, WA observes the current state of the environment. Based on this information, it chooses a suitable action called the proto action by using the actor-network. After having the proto action, WA searching for actions which are most relevant to the one. The action embedding layer is used to calculate the distance between all actions in action space compared to the proto action. Using the kNN algorithm [1], WA will select the most related k actions. Next, the information about the environment state and k-related actions will be used as an input for the critic network. At this point, instead of just evaluating the proto action, the critic network will evaluate k-related actions and the next state that is obtained after performing these actions. Finally, evaluation values calculated by the critic network will be used as a baseline to select the best action. The best action is chosen using the argmax function and will be used to interact with the environment at the current step.

The strength of this approach is instead of having to search randomly on the large action space, the agent can choose the best action among potential actions. Because the choice of action at each step is more oriented, the possibility that the agent chooses correct action is also larger. In this way, actions that have no value in solving the problem are likely to be rejected early on. As a result, WA helps the actor-critic algorithm better handle large action space environments. Besides, comparing proto action with k-related actions can also improve the performance of the algorithm. Using a larger set of actions for the critic network increases the likelihood of choosing the best action for the current state compared to just being able to select the proto action which seems right but not certain.

3.2 Multiple level action embedding

Although the effectiveness has been demonstrated, applying WA to problems is not an easy approach. One of the challenges when applying WA to any problem is building a proper embedding action. Initially, WA was only built to solve problems with discrete actions. Therefore, if operators want to apply WA for different problems, they need to find out how to perform all actions in the n-dimensional space. In a few cases, especially when there is no relationship between actions e.g. choosing between left or right actions, building an embedding can be difficult, not even possible.

When applying WA for PT, we are also facing the same problem. In PT environments, actions represent for attacker interactions with network systems. Since the attack is performed using complex approaches and technique, these actions are also dependent on many different factors. Therefore, the construction of embedding action is very difficult for PT.

The first thing that we have to determine during the construction of the action embedding for PT is which factors need to be considered. For example, we want to attack a laptop and to do this we need to scan the service running in the machine and then exploit a running service. With this problem, we see that two actions can be performed are "scan the machine" and "exploit a running service in the machine". If only based on the properties of the action ("scan" or "exploit") we can distinguish the actions from each other but cannot determine the relationship between them. However, if the object of the action, the machine, is included, the relationship between the actions becomes clear. Suppose, after scanning, there is more than one service running, we need to determine which service will be used to attack. Thus, to determine exactly an action we need to consider at least three factors: the characteristics of the action, the object of the action and the means of the action.

Based on the above idea, the paper proposes to build a multiple level action embedding using three levels including action characteristics, network structure and service vulnerabilities. Figure 2 and algorithm 1 show an overview of the embedding. We will discuss in detail these in the following sections.

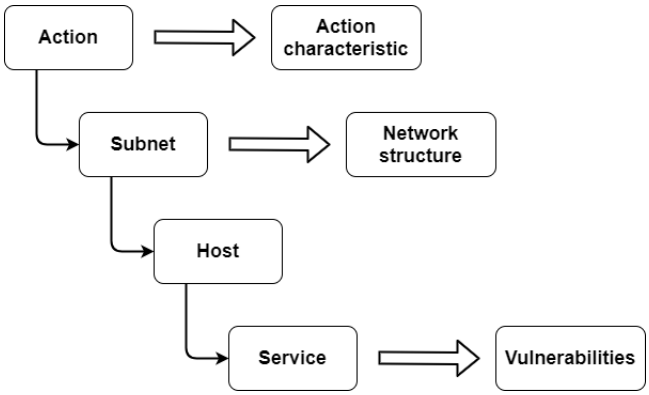


Figure 2: Multiple level action embedding

3.2.1 Action characteristics. The first level needs to be considered when building the embedding is the characteristic of the action. The action characteristic can be identified by many components

Algorithm 1: Multiple level action embedding implementation

Input: All actions in action space $a \in A$
Network structure $G(V, C)$; subnet $v \in V$; connection $c \in C$
All services $s \in S$ and its description $\{d_s, \forall s \in S\}$
Output: Vector representation $\{z_a, \forall a \in A\}$
Initialize Z empty;
for $a \in A$ **do**
  $z_a^1 \leftarrow a_{type}$;
  $z_a^2 \leftarrow a_{destination}$ {Unique ID for each action destination};
  $z_a^{3\rightarrow5} \leftarrow Node2Vec(G, window\_size)$ {Use Node2Vec to present network structure};
  $z_a^{6\rightarrow8} \leftarrow ServiceEmbedding(S, \{d_s \forall s \in S\})$ {Represent services based on its vulnerabilities' description};
  $Z.insert(z_a)$;
return Z;

such as their own action type and their destination (the object of the action).

First of all, an action can be defined by the type of action it belongs to. In real cyber-attacks, depending on the purpose of the attack, attackers have to perform a set of different actions. These actions can be divided into many different categories such as ping actions, scan actions, send actions and sniffing actions. Thus, the action type shows not only the properties of each activity but also the attack method or technique which contains the action.

In this work, the use of the network simulator, which will be introduced in an experimental setup, reduces the diversity of action categories. The simulator contains three basic types of actions including scan subnet, scan host and exploits services. In the action embedding, the first dimension is used to represent this information. In other words, the action embedding vector will use its first bit to describe these types of actions. For example, the first bit can be assigned -1 to represent the action type scan subnet, while scan hosts and exploit services actions are represented by the values 0 and 1, respectively. These values are designed to identify the correlation between actions. While scan subnet action and scan host action are in the group "scan", the similarity between them is greater than exploit services action. On the other hand, because scan host and exploit services action both have an impact on hosts, they are more interrelated.

Even though the first dimension can fully represent the properties of the actions on the network simulator, it is possible to clarify more clearly the difference between them by using the second dimension that focuses on the destination of the action. For scan subnet action, a subnet is considered as the destination of the action. While, for the other two actions (scan subnet and exploit services), the main destination of these actions is hosts. In the case that the number of hosts and subnets are fully acknowledged, these numbers can be used as values for the second bit of the action embedding vector. For example, the unique ID of each host and subnets can be used for the second bit. Next, these values are normalized to be in the range [-1, 1]. Using this technique is a possible approach to clearly define the different properties between similar types of actions. However, the downside of this approach is that the second dimension can not fully demonstrate the similarities between subnets and hosts since the unique ID is created randomly.

3.2.2 Network structure. As mentioned above, it takes more than the action characteristic level to fully show the relationship between the actions in the embedding. The second level shows this relationship in terms of representing the position of the action destination in the network. In other words, this level is built based on network structure.

Network structure represents the topology of the network. The essence of the network is the connection between the machines. When the number of computers in the network is large and for easy management, the machines are classified into different subnetworks. The subnets in the network are usually linked by network devices such as switches or routers. Because all hosts in a subnet are typically in a peer position and fully connected to each other, the network structure is largely determined by the way the subnets communicate with each other. In more detail, it describes the way the subnets are connected. Therefore, this article focuses on the subnet as the main component representing the network structure.

To illustrate the network structure, the paper used the Node2Vec model which is a graph embedding method [4]. Node2Vec is built to inherit the strengths of Random walk and Word2Vec [9]. It is capable of representing any graph as an n-dimensional vector but still retains the properties of the node in relation to its neighbours. Therefore, Node2Vec is a perfect fit for expressing network structure when our main concern is the relationships between subnets.

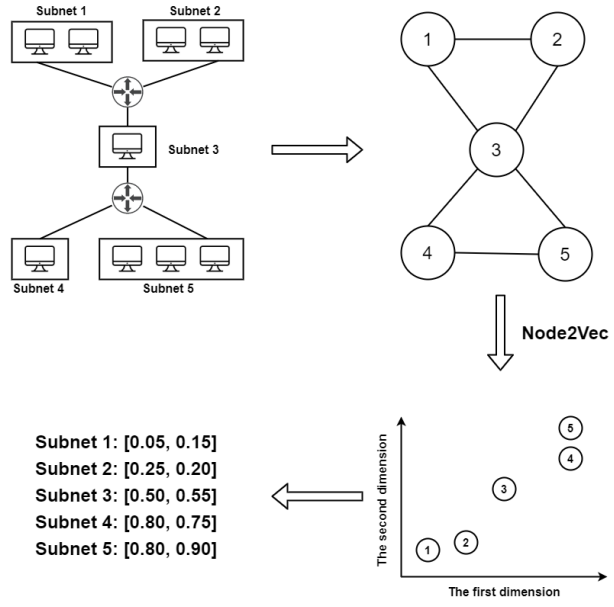


Figure 3: An example demonstrates network structure using Node2Vec on two dimensions space

Figure 3 shows results of representing network structure using Node2Vec on 2-dimensional space. Initially, all subnets that have the same connection to a network device such as a router are considered

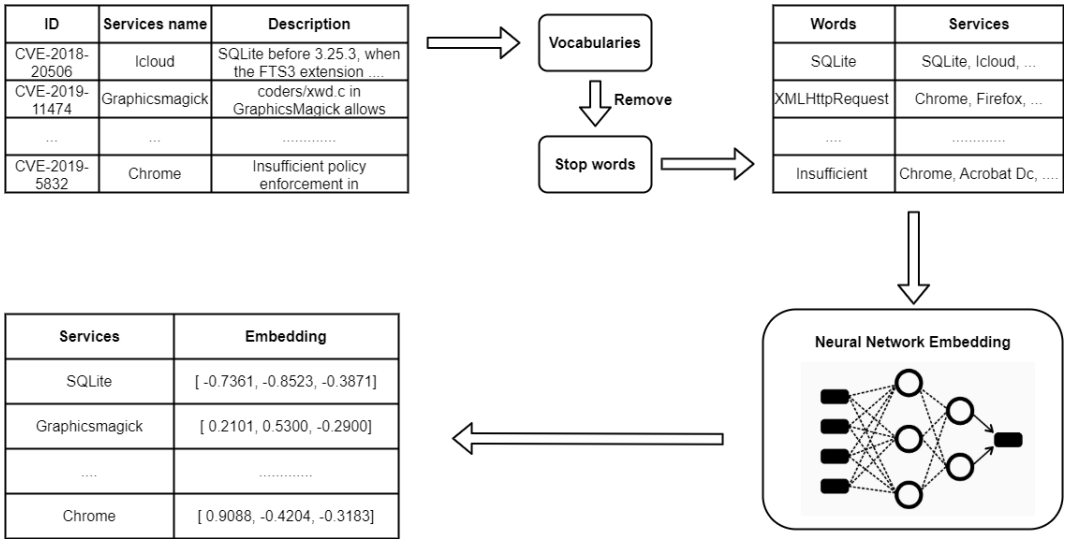


Figure 4: Services embedding

fully connected to each other. These subnets will be represented graphically as nodes linked together pair by pair. After having a full graph, Node2Vec is used to learn the features of the graph. Finally, the graph is demonstrated as an n-dimensional vector.

In fact, the paper converts the information about the subnet and connections into a three-dimensional vector instead. The reason behind this is making sure that the multiple level action embedding can deal with complex environments. At that time, the more information in the vector means the more accurate it is in representing the network topology. The information obtained from the process will be is described from 3^{th} to 5^{th} bits of the action embedding vector. It is worthwhile to notice that the network structure values each action received are based on the subnet that the action destination (the object of the action) belongs to. While a scan subnet action will obviously take value corresponding to the subnet, a scan host action and an exploit services action will be assigned the value of the subnet containing its host.

We will discuss in more detail how this level affects the performance of the RL. By giving the network topology to the agent, the agent is indirectly instructed to work in two directions: selecting a subnet that is directly related to the current attacked subnet and searching for potential sensitive hosts present in the current subnet. These directions contribute to increasing the strategy of the agent, thereby increasing the performance of the RL algorithm. For example, when the agent has successfully attacked a subnet, it will extend the attack to neighbouring subnets to increase the likelihood of success rather than searching in the entire network. In addition, since sensitive hosts are considered the primary target of an attack, it is appropriate to make the agent exploiting in deep into the current subnets to find sensitive hosts.

3.2.3 Services vulnerabilities. Having action characteristics and network structure levels, the action embedding is now able to distinguish actions and the object of each action in the network. However, when deploying exploits on hosts, services are also an important factor to be considered. In the network, because of containing many vulnerabilities, services are seen as a weaknesses point that attackers use to attack the network. Hence, the final level of the action embedding focuses on services vulnerabilities.

To determine which services are most likely to be successfully hacked and the relationship between the services, the article looks at security vulnerabilities that exist on the service. The idea is that the more vulnerabilities a service contains, the higher chance of it being attacked. In addition, the similarity between services can also be demonstrated by the similarity of the vulnerabilities they contain. However, information about the similarities of vulnerabilities was also not simple to obtain. So far, there are no studies focusing on this information.

Inspired by category embedding and word embedding, this work determines the correlation between the vulnerabilities using descriptive information about them. More specifically, all security vulnerabilities are introduced by its name, version and description on Common Vulnerabilities and Exposures details website [12]. This information source can be used to extract features through the neural network embedding layers. Each service contains all information describing the vulnerabilities it contains. From there, each service can be represented as an n-dimensional vector.

Figure 4 shows how the embedding works for services. Based on a service list containing existed services on the entire network, the vulnerabilities and its description related to these services will be retrieved. For example, Chrome service usually has vulnerabilities CVE-2019-5832 and CVE-2019-5831, so the descriptions of these vulnerabilities are taken into account. Next, a words list is constructed from this textual information. This word list contains the unique vocabulary in the data set as well as stop words such as "a", "an" and "the" have been removed from it. In other words, only vocabularies which contain important information describing how the vulnerabilities function are retained. This ensures the accuracy of the embedding after construction. After that, the pair set showing each relationship between the services and words are generated. It

will be used as input to the embedding network. Finally, the embedding network did the training process and results in the embedding vector for each service.

As mentioned above, a service may contain more than one vulnerabilities, so the embedding vector represents the service which will be the mean of all the vulnerabilities it contains. Likewise, for a host or a subnet, the embedding vector will be the mean of all the services contained in that host. This design is in line with the destination of the action types mentioned earlier. While the exploit service action should contain service information, scan host and scan subnet actions should contain embedding information of the host and the corresponding subnet.

4 EXPERIMENTS AND RESULTS

4.1 Experimental Setup

In this section, we will focus on the components used for experiments. there are three main components: the network simulator used to perform PT, experimental scenarios and evaluation metrics of experiments.

4.1.1 The network simulator. The network simulator is one of the important components of the system because it is the primary environment for performing PT. Although there are many methods that can be used such as real network and virtual machines, the network simulator is preferred because of its compactness and ability to support attacks. Currently, there are many network traffic simulators such as mininet and NS3, but these tools do not support network attack so it is difficult to use for PT. Therefore, the article builds the network simulator based on NAS a network simulator proposed by Schwart et al. [14]. It used text-based approach and was designed specifically for PT. As a result, NAS is not compact and can be applied to many different systems, but also fully supports attacks in PT in the environment with a large number of machines. However, the downside of NAS is that attacking a machine in the NAS is far more likely to succeed than it really is. Therefore, the article builds a NAS-based network simulator however adds more rules to increase the difficulty of the attack, while partially limiting the attacker's capabilities.

The network simulator consists of two main components including the network model and the MDP environment. The network model contains the core parts of the network and acts as a standalone system. It contains the basic components of the network simulator including subnets, connections, hosts, services and firewalls. The MDP environment used as agent input is defined by tuple including {State, Action, Reward, Transition}.

The components of the network model defined in the JSON file are then loaded and created the corresponding components. These components are built on the basis of NAS. While the connections, firewalls and services sections have only been changed in format, some rule changes are included in the subnets and hosts. Each component mimics the real-life network behaviour.

- A subnet contains one or multiple hosts.
- Connection determines which subnets are linked directly to each other.
- A host represents a device existed on the network.
- Services represent the applications running on a host

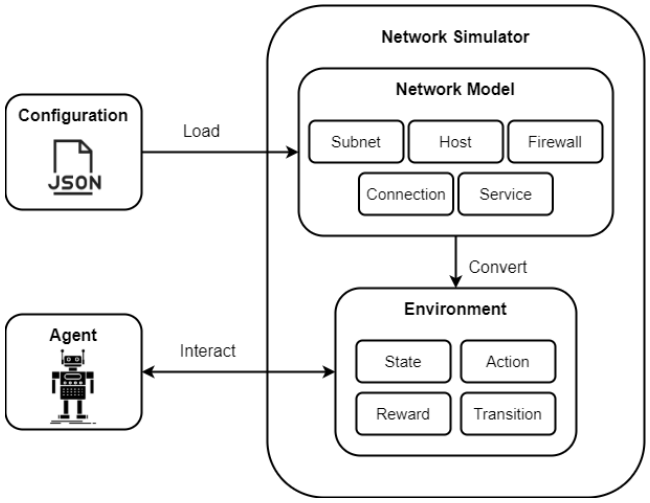


Figure 5: The Network Simulator

- Firewalls are used to limit the services that can be used to communicate between subnets.

In order for the RL algorithm to be able to solve the problems, they need to be put in a suitable format. In this article, PT problems are shown in the form of MDP represented by tuple {State, Action, Reward, Transition}. The state represents the knowledge that the attacker has in the current network environment. It is built based on the information from all the hosts including the scannable state, the attacked state, and the available state of each service running on all of the hosts. The action is defined to mimic the activities of real attackers. It includes three types of actions scan subnets, scan hosts and exploits service. The transition function shows how actions affect the environment. Finally, the reward function is used as the guideline for the agent. This function tells the agent when it has selected the correct action. To do that, the agent will receive an immediate reward for each corrective action. It will use this value to update its policy. In this article, the reward value is calculated by the cost for each action and the reward achieved for each action. While the cost of the scan action is equal value, the cost of the exploit service depends on each service. Every time the agent successfully attacks a normal it will get a reward value of r, for a sensitive host, the reward is 3r. This repeats continuously until the agent finds a way to achieve the maximum value in the environment.

4.1.2 Experimental scenarios. For the test involving the action embedding and the complexity of the environment, a standard network scenario is used. The standard scenario includes 5 subnets, 15 hosts and 5 exploitable services. The number of hosts in each subnet belongs to the set [3, 3, 2, 3, 4] respectively. Subnets are interconnected by two main clusters subnet (1, 2, 3) and (3, 4, 5). The number of sensitive hosts in the scenario is 5

For the experiment to test the scalability of the algorithm, a set of scenarios is defined. These scenarios have 5 subnets with the connection between subnets similar to the standard scenario. However, to test scalability, the number of hosts in these scenarios ranges from 10-256 hosts. On these scenarios, the number of sensitive hosts accounts for 10% of the total number of hosts. That

means scenarios with a large number of hosts not only have larger network sizes but also increase complexity.

4.1.3 Evaluation metrics. The multiple level action embedding is checked with logic accuracy. Although it is subjective, testing with this method still gives us an overview of the ability to represent actions of the embedding. For experiments applying the action embedding to WA, the proportion of sensitive hosts being successfully attacked is considered the primary metric. The reason for choosing this metric is to be consistent with the real-world attack process that sensitive hosts are positions that bring value to the attacker during the attack. The reward gained during the training is only used to check the ability of the agent to learn and ensure the results are not overfitting.

4.2 Experiments and Results

The effectiveness of multiple level action embedding is evaluated based on two main experiments: action embedding testing and the performance when applying it to WA for PT problems.

4.2.1 Action embedding testing. To evaluate the accuracy of the action embedding, some experiments were conducted. First of all, we conduct experiments on service vulnerabilities level. The purpose of the experiment is testing the ability of the level in classifying the vulnerabilities into the appropriate service category and determining the relationship between services. After that, we will test the accuracy of the multiple level action embedding by using it for some actions, analyzing the obtained result and ensuring that it satisfies logical accuracy.

To check the accuracy of the service vulnerabilities level, we tested on the top 45 applications ordered by the total number of distinct vulnerabilities from CVE details website. The description of these services is used as an input for the neural network embedding. After the training process, each service will be represented as a 3-dimensional vector.

Table 1: The top 5 most related services of Safari

Service	Similarity	Vector
Safari	1.000	[-0.272, -0.450, -0.843]
Icloud	0.998	[-0.319, -0.414, -0.807]
Reader	0.993	[-0.378, -0.416, -0.819]
Apple Tv	0.983	[-0.260, -0.594, -0.730]
Chrome	0.9704	[-0.470, -0.322, -0.794]

The experiment finding the top 5 most relevant services of the Safari application is shown in the table 1. The result shows that the applications similar to Safari are Icloud, Reader, Apple Tv and Chrome. Logically we can consider that this is an accurate result when Safari, Icloud and Apple Tv are all Apple apps, so the relationship between these services is undeniable. On the other hand, Safari and Chrome are both browsers so there are some similarities between them. With this result, we see that the service vulnerabilities level achieves its accuracy in representing the service to a vector-based on its vulnerabilities.

Table 2: Testing on different proto actions using multiple level action embedding

Proto aciton: Scan subnet 192.168.0.0

Action type	Desination	Serivce	Distance
Scan subnet	192.168.0.0		0.000
Scan host	192.168.0.1		1.014
Scan host	192.168.0.3		1.020
Scan host	192.168.0.2		1.030
Scan subnet	192.168.1.0		1.040
Exploit	192.168.0.1	Reader	2.030
Exploit	192.168.0.2	Reader	2.030
Exploit	192.168.0.3	Reader	2.030
Scan host	192.168.1.1		2.044
Exploit	192.168.0.1	Icloud	2.045

Proto aciton: Scan host 192.168.0.1

Action type	Desination	Serivce	Distance
Scan host	192.168.0.1		0.000
Scan host	192.168.0.3		0.043
Scan host	192.168.0.2		0.049
Exploit	192.168.0.1	Reader	1.000
Exploit	192.168.0.3	Reader	1.000
Exploit	192.168.0.2	Reader	1.000
Scan subnet	192.168.0.0		1.003
Scan host	192.168.1.1		1.004
Scan host	192.168.1.3		1.004
Exploit	192.168.0.1	Icloud	1.060

Proto aciton: Exploit service Safari on host 192.168.0.1

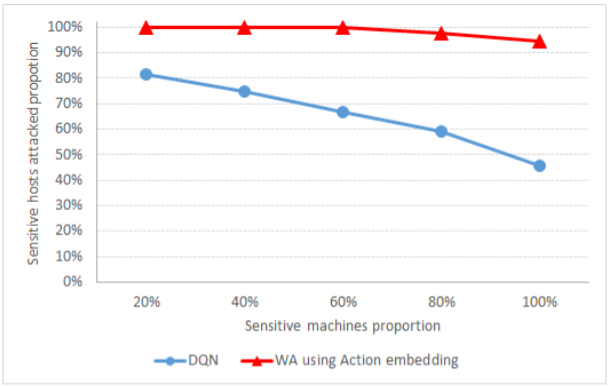
Action type	Desination	Serivce	Distance
Exploit	192.168.0.1	Safari	0.000
Exploit	192.168.0.3	Safari	0.000
Exploit	192.168.0.2	Safari	0.000
Exploit	192.168.0.2	Icloud	0.069
Exploit	192.168.0.1	Icloud	0.069
Exploit	192.168.0.3	Icloud	0.069
Exploit	192.168.0.3	Reader	0.114
Exploit	192.168.0.1	Reader	0.114
Exploit	192.168.0.2	Reader	0.114
Exploit	192.168.0.2	Apple Tv	0.183

Next, we perform an experiment to see the accuracy of the embedding for actions. Table 2 shows us the top 10 actions that are most similar to some proto actions including three different types of action. For the proto action "scan subnet 192.168.0.0", the most related actions is scan hosts existed in the subnet 192.168.0.0 and scan the subnet 192.168.1.0 which is directly connected to it. This result demonstrates that the embedding increases the agent's strategy by finding potential hosts on the current subnet as well as expanding the search for neighbouring subnets. For the proto action "scan host 192.168.0.1", the embedding gives the same results. However, since the proto action is scan host, the order of the agent's tactics has changed. At this point, the agent will prioritize scanning all hosts on the network, then try to see if it can exploit those hosts and finally scan the subnet that contains the current host. This is a sensible tactic that helps agents increase their likelihood of success. Finally, for the proto action "exploit service Safari on host 192.168.0.1", the agent prioritizes the exploit action type and focus on service that most related with Safari on all of the hosts on current subnet. With this strategy, the agent can check which service is suitable to attack the 192.168.0.1 host and check if other hosts can be hacked with the Safari service. Thus, the agent's ability to attack successfully is also increased. In conclusion, we find that regardless of the proto action, the multiple level action embedding enhances the agent's ability to attack success by indirectly increasing its tactics.

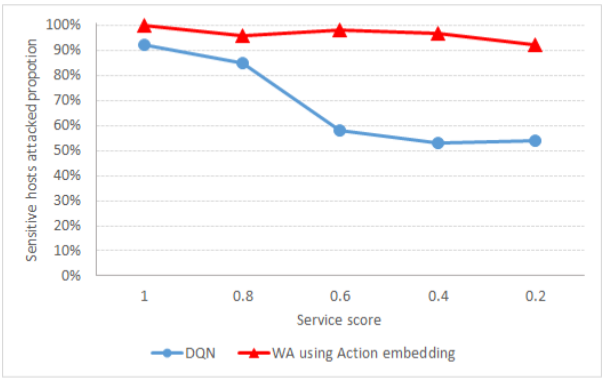
4.2.2 Evaluation of application effectiveness and WA. The complexity of the environment is represented by three configurations: sensitive host configuration, service score configuration and quantity configuration. The sensitive host configuration is about the ratio of the number of sensitive hosts to the total number of machines. The higher this ratio, the more complicated the problem. The service score configuration is about the rate of attack success when using certain service. The lower the service score means the harder it is for the service to be hacked, which means, the higher the complexity of PT problems. Quantity configuration is about the total number of machines on the network. The larger the number of machines, the harder it is for the agent to attack successfully. To evaluate the effect of the embedding on the RL for the PT problem, we perform the experiment on all three configurations. The complexity of the configurations in the experiments will be increased gradually.

All experiments related to sensitive host and service score configurations will be done on an environment with 15 hosts and 5 services. In each episode, the number of max steps the agent can execute is twice the action space. The test will run in a total of 1000 episodes. For sensitive host configuration, the sensitive host ratios performed in the experiment are 20, 40, 60, 80 and 100%, respectively. For service score configuration, attack success rates of all services will decrease in the order of 100%, 80%, 60%, 40% and 20%. On the other hand, for quantity configuration, we will conduct experiments with network size in a range of 10-256 hosts.

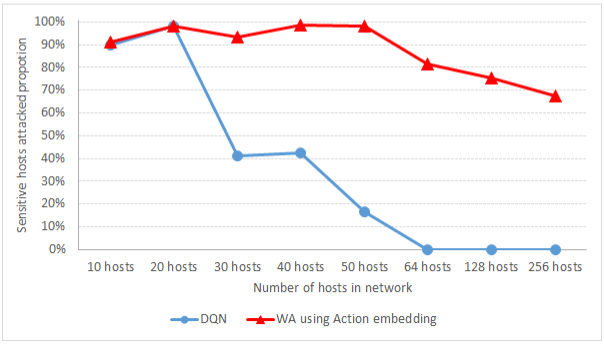
The figure 6 shows the results of these experiments. With the changing in the sensitive host configuration (figure 6a), as the rate of sensitive hosts increases, the performance of the DQN algorithm decreases from 81% to 46%, corresponding to the sensitive host ratio of 0.2 and 1. Meanwhile, WA using the multiple level action embedding performances remained at greater than 95%. For the



(a) Sensitive host configuration



(b) Service score configuration



(c) Quantity configuration

Figure 6: The sensitive machine attacked proportion results of three configurations

service score configuration, experimental results recorded the same pattern (figure 6b). The DQN algorithm is greatly affected by the service score, performance drops from 90% with service score of 1 to nearly 50% when service score is 0.2. In contrast, the WA using the embedding was only slightly affected by the service score and kept performance greater than 90%. In addition, although the DQN algorithm has very good performance with networks with a small

number of hosts, with a number of machines greater than 64 the algorithm is no longer capable of solving the problem and hitting the bottom 0% (figure 6c). Although the WA using the action embedding performance is still influenced by the number of machines, on the large network with 256 hosts, it still keeps an acceptable performance around 70%. These results prove that applying the multiple level action embedding with WA into the PT problem is extremely efficient. It not only increases the network size that the algorithm can solve but also increases the accuracy of the algorithm when facing complex environments.

5 CONCLUSION

In this work, we focus on increasing the applicability of RL when applying to PT. The multiple level action embedding is proposed to increased the accuracy and performance of RL algorithms when solving PT problems. It is also an important component for applying to the Wolpertinger architecture which is an effective method for environments having large action space. Many experiments have been conducted to evaluate the accuracy of the action embedding. DQL algorithm is also used as a baseline to compare with WA using the action embedding.

The main contribution of the paper is to improve the efficiency of the RL to solve the PT problem when the network has a large number of hosts and a complicated environment. Using multiple level system including three levels: action characteristics, network structure and services vulnerabilities; the action embedding is capable of accurately representing actions in the form of n-dimension vectors, distinguishing and calculating the relationship between them. As a result, the multiple level action embedding can instruct the agent indirectly to increase its tactics during the training process.

The result shows that the embedding not only is able to represent all actions in the environment's action space but also find all actions most relevant to any given one while still being logically accurate. When applied to the RL for PT, the WA using the action embedding outperforms the DQN algorithm for complex environments. Even with the high sensitive hosts rate (up to 100%) and the low service score (service success rate = 20%), it still keeps performance above 90%. Additionally, the network size of up to 256 hosts can still be handled with an acceptable accuracy of about 70%.

The next development direction of the problem might be to apply it to more realistic environments in order to reduce the distance between this research and the real-world network system. Additionally, using the word embedding to show the correlation between the problems could cause confusion if the description of the vulnerabilities was incomplete. Hence, a method which is able to show the correlation between services based on their properties could be developed to deal with this situation.

REFERENCES

[1] Sahibsingh A Dudani. 1976. The distance-weighted k-nearest-neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics* 4 (1976), 325–327.

[2] Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. 2015. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679* (2015).

[3] Mohamed C Ghanem and Thomas M Chen. 2020. Reinforcement learning for efficient network penetration testing. *Information* 11, 1 (2020), 6.

[4] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 855–864.

[5] Hado V Hasselt. 2010. Double Q-learning. In *Advances in neural information processing systems*. 2613–2621.

[6] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. 1996. Reinforcement learning: A survey. *Journal of artificial intelligence research* 4 (1996), 237–285.

[7] Xiaoyuan Liang, Xunsheng Du, Guiling Wang, and Zhu Han. 2018. Deep reinforcement learning for traffic light control in vehicular networks. *arXiv preprint arXiv:1803.11115* (2018).

[8] Long-Ji Lin. 1992. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning* 8, 3-4 (1992), 293–321.

[9] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.

[10] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*. 1928–1937.

[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[12] S Ozkan. 2011. CVE Details: The ultimate security vulnerability datasource. Retrieved March 20, 2020 from http://www.cvedetails.com

[13] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. 2015. Prioritized experience replay. *arXiv preprint arXiv:1511.05952* (2015).

[14] Jonathon Schwartz and Hanna Kurniawati. 2019. Autonomous penetration testing using reinforcement learning. *arXiv preprint arXiv:1905.05965* (2019).

[15] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. 2016. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*. 1995–2003.

[16] Zhiheng Zhao, Yi Liang, and Xiaoming Jin. 2018. Handling large-scale action space in deep Q network. In *2018 International Conference on Artificial Intelligence and Big Data (ICAIBD)*. IEEE, 93–96.