Applying reinforcement learning in automated penetration testing

Nguyen Thanh Cong, Nguyen Viet Hung

***Abstract*— Facing increasingly diverse and frequent information security threats today, penetration testing is a security assessment method for information systems that organizations prioritize. Pentesters usually perform penetration testing manually and can detect critical bugs and information security issues. However, this method requires much work and requires pentesters to have high levels of practical experience and qualifications. One of the current research directions that has been interested recently is methods to support automated penetration testing. Several research groups have used attack graph analysis techniques and reinforcement learning algorithms worldwide to make automated pentesting tools. This paper proposes a model based on a reinforcement learning algorithm and parameter optimization method for this model in automated pentesting problems. To evaluate the proposed model, we utilize the data set based on the method used by other research groups. We also assess the self-built dataset on real environments with vulnerabilities. The experimental results show that the proposed method gives better assessments than other methods.**

***Tóm tắt*— Đối mặt với những nguy cơ mất an toàn thông tin ngày càng đa dạng và thường xuyên hiện nay, kiểm thử xâm nhập là một phương pháp đánh giá an toàn cho các hệ thống thông tin được các tổ chức ưu tiên thực hiện. Kiểm thử xâm nhập thường được thực hiện thủ công bởi các kiểm thử viên và có thể phát hiện những lỗi nghiêm trọng và các vấn đề về an toàn thông tin. Tuy nhiên, phương pháp này đòi hỏi nhiều công sức, đồng thời yêu cầu kiểm thử viên phải có trình độ, kinh nghiệm thực tế cao. Một trong những hướng nghiên cứu được quan tâm gần đây là xây dựng các phương pháp hỗ trợ đánh giá kiểm thử tự động. Kỹ thuật phân tích đồ thị tấn công và thuật toán học tăng cường đã được một số nhóm nghiên cứu trên thế giới sử dụng trong xây dựng công cụ đánh giá kiểm thử tự động. Trong bài báo này, nhóm tác giả đề xuất mô hình dựa trên thuật toán học tăng cường và phương pháp tối ưu hóa tham số cho mô hình này trong đánh giá kiểm thử tự động. Để đánh giá mô hình đề xuất, ngoài bộ dữ liệu dựa trên phương pháp được các nhóm nghiên cứu khác sử dụng, chúng tôi còn đánh giá trên bộ dữ liệu (dataset) tự xây dựng trên các môi trường thực tế với các lỗ hổng bảo mật cập nhật. Các kết quả thực nghiệm cho thấy phương pháp đề xuất đưa ra các đánh giá tốt hơn các phương pháp khác.**

***Keywords*— Penetration testing; reinforcement learning; information secutity.**

***Từ khóa*— Kiểm thử xâm nhập; học tăng cường; an toàn thông tin.**

I. INTRODUCTION

Penetration testing is assessing an information system's security through actual attacks conducted by security experts (pentesters). In other words, pentesters try to attack the system in the same way as hackers do to find the system's vulnerabilities. Through the results of such attacks, pentesters can detect actual weaknesses and accurately assess the system's security. Because of its effectiveness in practice, penetration testing gradually becomes mandatory and is performed periodically for information systems that require a high level of security.

There are three types of penetration testing: black box testing, gray box testing, and white box testing [1, 2]. In gray box testing, pentesters are provided with little information about the system, such as IP address and network architecture [2], instead of having no information about the system as in black box testing or all information in white box testing [2]. Depending on the security assessment requirements, owners of information systems

This manuscript is received on October 27, 2022. It is commented on November 08, 2022 and is accepted on November 14, 2022 by the first reviewer. It is commented on December 02, 2022 and is accepted on December 16, 2022 by the second reviewer.

and testers can choose the appropriate type of testing.

Qualified pentesters usually perform penetration testing, usually experienced security professionals skilled in attacking information systems (white hat hackers). This complex process requires a lot of time and effort to maintain. Therefore, many recent studies have focused on testing methods that enable automated penetration testing support.

One of the critical tasks of automated penetration testing methods is to model the components in the penetration testing environment, such as the network model, network devices, network vulnerabilities, and attack paths. Attack graph analysis is one of the most widely used techniques for system modeling in automated penetration testing. An attack graph is a model that describes all possible network attack paths in a network system, where the end node will be the state in which the attacker has successfully attacked the predefined target. Expressly, the nodes modeled in the attack graph represent the network state, such as vulnerabilities in devices, connections between devices, and services running on devices. In addition, the attack graph has several logical nodes representing the rule and the subsequent network state inferred from the initial states and that rule. Based on these logical nodes, the attack graph can check whether the initial states can lead to compromised states or attacks on a network system device. Besides, the attack graph is a directed graph, and the directed edges in this type of graph represent the attack progress or state transition of the system intruder through the network state nodes. When two state nodes are connected, an attacker can move from the original state node to the destination state node.

Conversely, if there is no connection between two nodes, the attacker cannot transition from one state to another. From there, the set of connected state nodes that result in the final state being an undesired state, such as being attacked by a device in the system, provides a network attack path to that undesired state.

Thus, using an attack graph, we can represent and evaluate the possible paths taken by an attacker to reach a specific target in the network.

Automated penetration testing is the process of automating the search for attack paths from a particular node (the pentester's machine) to the target node (device or system needed to be pentested) given [3, 4]. In reality, network systems can have many different attack paths. Finding the optimal attack path in the attack graph can be very complex based on the number of devices in the system and the number of vulnerabilities on these devices. Many methods have been used to find the optimal attack path, such as graph analysis, depth-first search, and breadth-first search. However, because the attack graph modeled the existing network system with many complex parameters, dynamically dependent on the state of all the system’s components, searching for the static graph revealed many problems and limitations. One of the research directions that has proven to be effective in finding the optimal attack path is the method of Reinforcement Learning. Reinforcement Learning allows the environment's responses to be assessed by actors when specific actions are performed. The goal of reinforcement learning is to find the best policy so that the agent gets the best results in each action. Some studies have used reinforcement learning for attack graph analysis. However, the performance of reinforcement learning algorithms depends a lot on solving the problem of balancing the exploration-exploitation ratio between actions and focusing on the best action (exploration-exploitation dilemma). Therefore, with random or non-evaluated parameters, the performance of these models could be non-stable and optimal.

In this article, our contributions include the following:

1. Proposing principles for building the state and actions of the agent in the reinforcement learning model for attack graph analysis and automated testing of computer systems.

2. Proposed methods contain optimized parameters and evaluate reinforcement learning

models through automatically generated datasets tested in a real environment.

The rest of the article is arranged as follows: In Chapter II, we present and evaluate some studies on finding the system's optimal attack path. Chapter III explains the attack graph, the attack graph generation process, and some related tools. In chapter IV, we present a reinforcement learning model based on the Q-Learning algorithm, problems in applying reinforcement learning algorithms, and solutions for those problems. Chapter V presents a proposed parameter optimization method for reinforcement learning models. Finally, in chapter VI, we present the automatic method of generating labeled datasets and the evaluation and test results of the model with the auto-generated dataset and the actual network model.

II. RELATED WORK

Many models have been used to identify network attacks, such as attack trees, Petri networks, and attack graphs. The attack graph was first proposed by Swiler et al. [5] in 1997 to describe the state of network attacks. Attack graphs are used to determine whether or not an attacker can achieve a predetermined network state by attacking based on the current system's network state [4]. Using the Markov Decision Process is one of the attack graph analysis methods. An attack graph often includes many attack paths or ways of reaching a target network state. The attacker will manage to choose the attack path so that the effort to achieve it is minimal and the reward for the attack path is the largest. The Markov Decision Process can determine the best-evaluated set of actions from a sequence of random actions. In the study [6], Sheyner proposed for the first time to calculate the best attack path based on the Markov Decision Process model.

The method used is Value Iteration to find the optimal attack path. Value iteration algorithms often take a long time for the values to converge. In addition, it depends on the probabilities between actions that are generally unknown to the agent before taking the actions. Therefore, evaluating all cases of an extensive network or a large number of security vulnerabilities on network devices can be challenging. However, the use of reinforcement learning algorithms based on the Markov decision process model has been studied and experimented with in the hope that it is possible to reduce computational cases with complex attack graphs. In [7], the reinforcement learning model was first proposed for automation pentesting. Applying reinforcement learning to the attack graph analysis process can reduce the times a graph's branches must be traversed. Since then, the time to analyze the attack graph has also decreased. When reinforcement learning is applied, the attack graph is modeled as an environment. It allows the agent to act as a pentester and be trained in this environment. Typically, network states are defined by the nodes on the graph, and actions are represented by branches associated with the corresponding nodes. However, the current model mainly uses reinforcement learning algorithms with unoptimized parameters and no specific performance evaluation. It makes the model still limited and challenging to apply in testing problems with particular network systems.

In [8], the authors used a reinforcement learning model to search for exploit modules in the Metasploit Framework. However, the input information of the learning model is relatively simple reconnaissance information, and the model has not yet handled the dynamic states of the environment.

In [7], a deep reinforcement learning model is proposed to apply to the automated testing problem. However, to increase the performance of the deep reinforcement learning model, the author introduced the concept of a simplified matrix containing all attack paths. This matrix is generated from the original score matrix using a depth-first search algorithm. This limits the agent to pre-calculated attack paths and makes it difficult to update to the real environment. The main contribution of this research is the use of practical tools to apply to the system allowing testing with real systems. In addition, the model is still limited to the trained network and has not been applied to the new network model. This

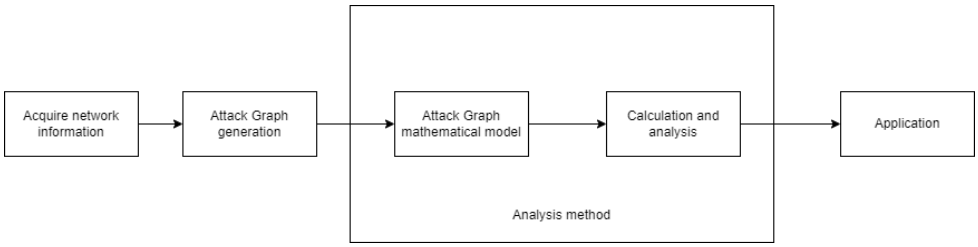


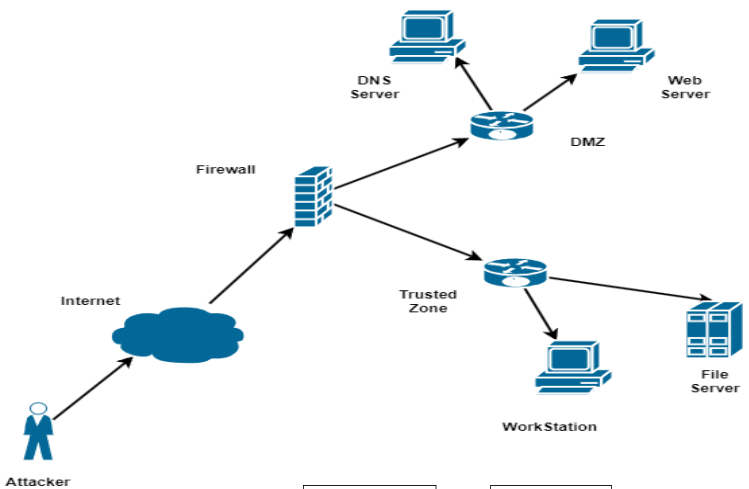
Figure 1. Generation and application of attack graph [19]

causes the deep reinforcement learning model to be less flexible in real-world conditions.

One of the problems with reinforcement learning models is balancing between discovering other paths and selecting the current best ones. In addition, the algorithm and parameters must be chosen objectively to optimize the results' accuracy. Therefore, to solve this problem, we propose applying the optimized parameters for the reinforcement learning model and automatically creating a network topology dataset with the optimal attack path as the label suited for penetration testing. In addition, we proposed to develop the environment and methods for applying reinforcement learning to the network penetration testing problem in reality.

III. BACKGROUND

The attack graph is one of the crucial components in the penetration testing automation process. The attack graph includes all information about the attack paths to the target in the network and is the basis for performing a network security analysis of a system. Therefore, creating an attack graph is essential and significantly influences the system's automated pentesting process. The generation of the attack graph could be done in several steps. In this section, we introduce the process by which we create the attack graph



Device	*Description*	*CVE-ID*
Web Server	Allow remote execution of code	CVE-2015-1635
WorkStation	Allow remote execution of code	CVE-2013-4465

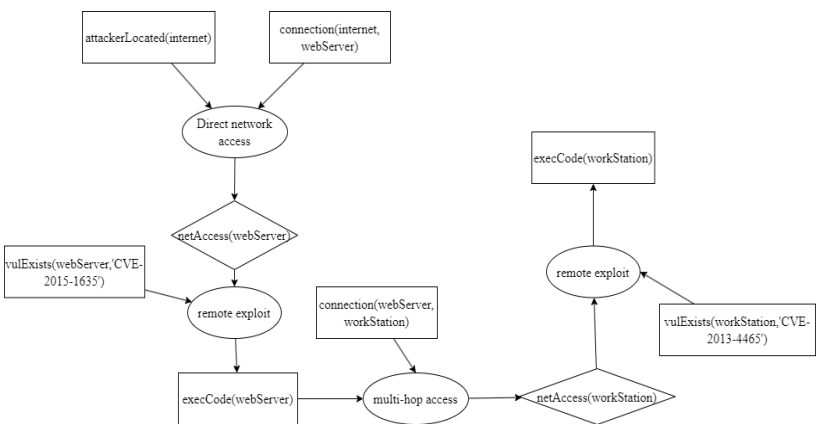


Figure 2. Example of a network system and attack graph [19]

and the technologies in that process. In the first step, we use the OpenVas tool to scan the system for vulnerabilities. Then we use the MulVAL tool to build the attack graph. To be able to apply the reinforcement learning algorithm, we build the environment and define the reward points for actions that correspond to the state of the agent. Finally, we use the Q-learning algorithm to train the agent to find the optimal path to reach the goal.

A. Attack Graph Creation and Application Process

The process of creating and using an attack graph is shown generally in Figure . In the first stage of attack graph generation, information about the network must be collected first. This information includes the network topology, vulnerabilities in the system, and network configuration. After gathering information about the network system, this information is analyzed by the analysis system. After that, it would be applied to specific applications.

The attack graph is created to represent the general state of the network's security using a directed graph. As mentioned in Chapter II, the nodes modeled in the attack graph represent the network state, such as vulnerabilities in devices, connections between devices, and services running on devices. The nodes could also represent the logical rule and the result obtained from that rule. Meanwhile, the directed edges of the graph represent that an attacker can transition from one network state to another.

In the example of a network topology shown in Figure 2, the network model consists of three network partitions: Internet zone, DMZ (demilitarized zone), and intranet zone. The information about the vulnerabilities collected on the corresponding network system is shown in the table in Figure 2. The generated attack graph with information about the network topology and the vulnerabilities above is shown in Figure 2. Then, the attack graph depicting the process of the attacker's intrusion attack could be performed according to the following steps:

- The attacker connects from the Internet to the webserver.
- The attacker compromises the web server with the vulnerability CVE-2015-1635.
- From the compromised web server, the attacker connects to the workStation in the Trust Zone.
    - The attacker compromises the workStation with vulnerability CVE-2013-4465.

B. Vulnerability scanning using OpenVAS

To collect information about the vulnerabilities of devices in the network, we use the OpenVAS tool. OpenVAS (Open Vulnerability Assessment System) is an open-source vulnerability scanning tool developed by Greenbone Network. The OpenVAS engine includes a large number of scanning modules and supports a variety of frequently updated scanning modules [9]. In addition, OpenVAS also supports making API calls in the Python programming language, allowing programmers to automate scanning through programming code. Therefore, with these features, OpenVAS is the right tool for us to quickly automate vulnerability scanning and extract information from the results of received notifications to build attack graphs.

C. MulVAL

After having information about the network's vulnerabilities, we used the MulVAL tool to create an attack graph. MulVAL is an open-source attack graph analysis tool developed by a research group at the University of Kaiserslautern. MulVAL supports the analysis of attack graphs according to the attack model of the US government's security policy (US-CERT) [10]. MulVAL allows us to analyze vulnerabilities on a network logically. MulVAL uses Datalog language to model the entities and security state of the network during analysis, such as vulnerabilities and connections between devices.

Datalog is a declarative logic programming language commonly used in database queries. However, in recent years, Datalog has also been used in data extraction, program

analysis, and security applications. Datalog analyzes information through premises and defined rules [11]. Here are some examples:

- Ancestor(X, Y):- parent(X, Y).
- Ancestor(X, Y):- parent(X, Z), ancestor(Z, Y).

In the above example, the above two clauses mean: Entity X is an ancestor of Y if X is the parent of Y. Entity X is an ancestor of Y if X is the parent of Z and Z is an ancestor of Y.

With the same reasoning, the Datalog language can logically analyze premises and deduce possible outcomes. In addition, MulVAL contains pre-existing network security-related rules that can be analyzed from the input premises. Once the prerequisite information is provided, the analysis can be performed in seconds with thousands of computing devices [12, 13]. For this reason, the MulVAL combined with specific scanning tools such as OpenVAS can quickly help us automate the construction of attack graphs. Figure 2 is an example of an attack graph generated by the MulVAL tool. This attack graph comes from the node attackLocated (Internet), which represents the current state of the attacker's location (attackerLocated). Based on network states existing on the systems, such as vulnerabilities (vulExists), network services (networkServiceInfo), and network connections between devices (hacl), the attacker makes the transitions to reach the final network state, which compromises the workStation on the system at the execCode(workStation) node.

IV. Reinforcement learning model in automated penetration testing

The reinforcement learning model is one of the most commonly used machine learning models. Reinforcement learning consists of environmental, state, and action components. The environment is modeled as a Markov process. Each state of the Markov process is a state of the environment. In this section, we will explain the Markov Decision Process, the reinforcement learning algorithm used in the model, the process of building an environment for the reinforcement learning model, and the use of the model applied to the automated penetration testing problem.

A. Markov Decision Process

The Markov Decision Process (MDP) is a framework that allows decision-making modeling in specific situations where the outcome is partly random and partly under the control of a decision-maker. MDPs were first known in the 1950s. They are used in various fields, including robotics, automatic control, economics, and manufacturing.

The MDP uses a set of 5 values (S, A, P, R, γ) to describe a decision process. S stands for the set of possible states; A stands for the set of actions; P stands for the state migration matrix; R is the result received after changing the state by a specific action; and γ is the discount factor to express uncertainty about the future.

At the beginning of the MDP, the agent comes from a specific state and, based on the probability, is determined the possible actions corresponding to the state, which are determined by the value P (s'|s, a) to move to state s'. An example of a Markov decision process is shown in Figure 3.

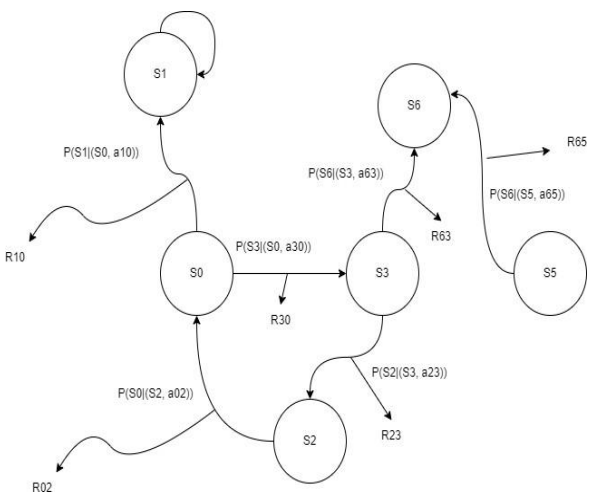


Figure 3. An example of Markov decision process

B. Reinforcement Learning Model

As mentioned above, reinforcement learning is about finding optimal actions to achieve the

ultimate goal for states. This process is summarized as shown in Figure 4.

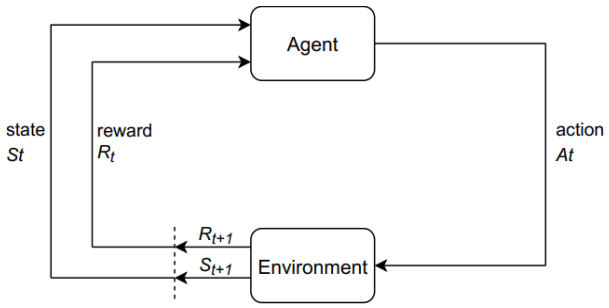


Figure 4. Reinforcement Learning Process

Reinforcement learning learns to map the current state to the subsequent corresponding actions. The agent will have no idea what action to take and will have to learn and discover which subsequent action best corresponds to the current state by attempting those actions. One of the most exciting and challenging problems with reinforcement learning is that actions can affect the immediate reward and subsequent states and rewards. Therefore, two prominent features of reinforcement learning are trial-and-error and delay scores that distinguish them from other types of machine learning.

In general, the reinforcement learning model can be described as follows:

1. The agent is put into any state.
2. The agent acts and receives a bonus point.
3. The agent updates the action value corresponding to the current state.
4. The agent transitions to the new state and returns to step 2.

In reinforcement learning, Q-Learning and Sarsa are two algorithms that represent two types of on-policy and off-policy reinforcement learning algorithms.

The Sarsa model interacts with the environment and updates the policy based on its actions. Therefore, Sarsa is considered an on-policy reinforcement learning algorithm. The Q value for the state-action pair is updated through trial and error, which is scaled by the alpha learning rate. The Q values represent the number of rewards the model gets if it takes that action in the next step. The update function of the algorithm is shown below:

$$Q(s_t,a_t)=Q(s_t,a_t)+\alpha[r_t+\gamma*Q(s_{t+1},a_{t+1})-Q(s_t,a_t)] \quad (1)$$

In there:

- $Q(s_t,a_t)$ is the Q value of the state s_t and the action a_t.
- α is the learning rate.
- r_t is the reward.
- γ is the discount rate.
- $Q(s_{t+1},a_{t+1})$ is the Q value of state s_{t+1} and action a_{t+1}.

Q-learning is a reinforcement learning algorithm quite similar to Sarsa. However, unlike Sarsa, Q-learning evaluates a pair of action states based on the maximum potential it can get in the next state and does not need to follow a continuous process.

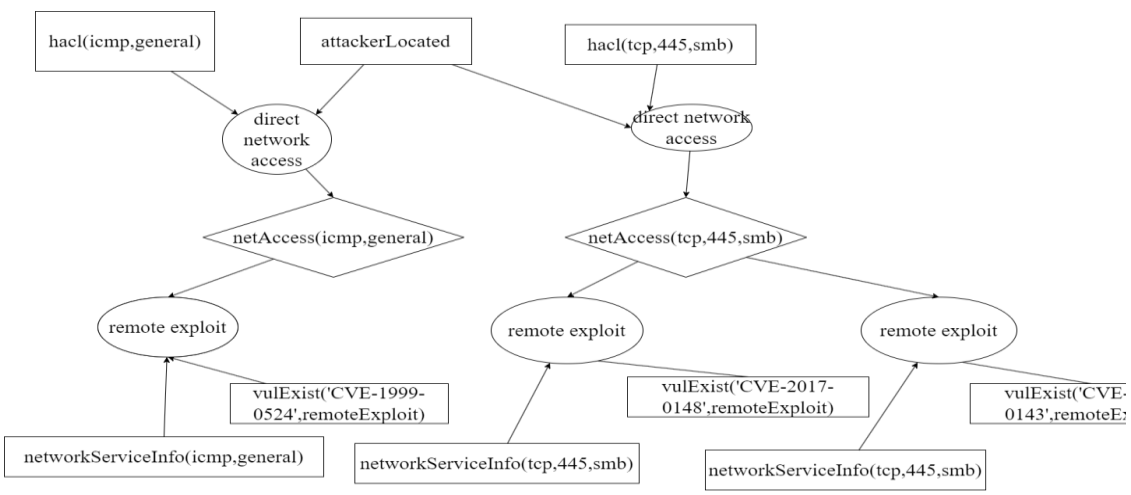


Figure 5. An example of an attack graph

$$Q(s_t,a_t)=Q(s_t,a_t)+\alpha\left[\begin{matrix}r_t+\gamma*\max Q(s_{t+1},a_{t+1})\\ -Q(s_t,a_t)\end{matrix}\right] \quad (2)$$

In there:

- $Q(s_t,a_t)$ is the Q value of the state s_t and the action a_t.
- α is the learning rate.
- r_t is the reward.
- γ is the discount rate.
- $\max Q(s_{t+1},a_{t+1})$ is the maximum Q value of state s_{t+1} and action a_{t+1}.

The attack graph is examined in chronological order, beginning with the connection and ending with the vulnerability. Usually, actions in a state are initialized in equal proportions. Therefore, if the number of vulnerabilities on a connection port is significant, the number of attempts per vulnerability on that port will decrease. It means that the reliability of those vulnerabilities will be reduced. To solve this problem, instead of using the original Epsilon-Greedy algorithm, the connection service selection step will be allocated probability according to the number of subsequent vulnerabilities it has.

$$P(s_t, a_t)=\frac{\text{Number of vulnerabilities if choose the following n}}{\text{umber of vulnerabilities in the next steps}} \quad (3)$$

For example, in Figure 5, the number of vulnerabilities for choosing to go through netAccess(ICMP, general) is two. If choosing to go through smb, the number of vulnerabilities is four, then the probability that the agent chooses ICMP is 1/3 and smb is 2/3.

With this problem, the Sarsa algorithm proved to have more weaknesses than the Q-Learning algorithm. The reinforcement learning algorithm trains the model by trying actions and getting rewards. For state-action pairs, the more actions that are performed, the higher the confidence in that pair. Suppose we use an on-policy reinforcement learning algorithm like Sarsa. In that case, we will encounter an issue where the traversal to the node presenting the connections will depend on all the vulnerabilities it has on that service port. This is in contrast to the fact that most of the vulnerabilities will be evaluated independently with the connections. The differences between Q learning and the Sarsa algorithm in penetration testing are shown in Table 1.

TABLE 1. DIFFERENCES BETWEEN Q LEARNING AND THE SARSA ALGORITHM IN PENETRATION TESTING

Q learning (off-policy)	Sarsa (on-policy)
Service selection depends on the maximum Q value of the following states	Service selection depends on all Q values of the following states
Only the maximum return's vulnerability exploitation affects service selection	More failed exploitation or less return of any vulnerabilities on service affect service selection
More suitable for real penetration testing	Less suitable for real penetration testing

In the case in Figure 5, the agent has to choose between moving through ICMP or TCP on port 445. The Sarsa algorithm is an on-policy reinforcement learning algorithm, so the service selection step will depend on the agent's choice of two vulnerabilities in a later step. As a result, the next step's vulnerability selection will not be evaluated relatively. In particular, it is more beneficial to choose a service with fewer vulnerabilities if it is to initialize the Q values at 0.

For the on-policy algorithm, the score of the previous action will be updated according to the highest number of points the agent can get for the following action. Therefore, in the above case, access through the service port will not depend on all its vulnerabilities but only on the vulnerability with the highest score existing on the service port. Therefore, applying an on-policy algorithm like Q-learning to this problem will avoid the above case.

C. Building an environment for reinforcement learning

We need an environment for the training model to apply the reinforcement learning algorithm. Therefore, to evaluate network

vulnerabilities against each other, we need a basis to evaluate these vulnerabilities. To assess vulnerabilities, we use the CVE NVD database.

1. CVE NVD Database

To apply reinforcement learning algorithms, we must evaluate the agent's states and actions. NVD (National Vulnerability Database) is a US government standard repository for vulnerability data management.

This repository allows easy management and assessment of vulnerable data. This database includes almost all vulnerabilities, such as software and configuration errors. Vulnerabilities in the database are collected from 1988 to the present. The CVE NVD data includes detailed score parameters, including the Base Score or overall score, the Impact Score, and the Exploitability Score, as shown in Figure 6. The penetration score measures the likelihood that the vulnerability can be exploited to compromise the device. Meanwhile, the impact score assesses the harmful level of the vulnerability when exploited, and the base score represents the vulnerability in the general case. By utilizing these parameters, vulnerabilities are evaluated and can be compared between vulnerabilities.

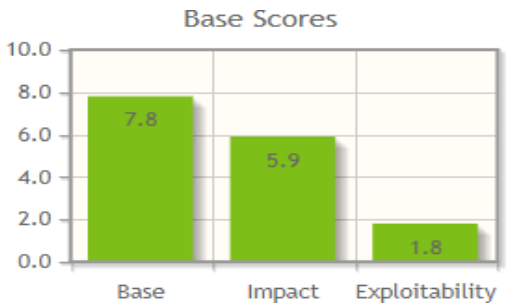


Figure 6. Vulnerability Score in NVD

2. Definition of the state and action of the agent in the reinforcement learning model

To model the environment for reinforcement learning, we need to define the state and actions of the agent for the network penetration testing problem. In research [7] and [14], the author uses the node itself for the state, and the corresponding actions are the edges connected to that node. In Figure 7, the green node is defined as the node from which the agent initiates; the red nodes are the targets to which the agent has to go. The agent can go from the starting position to the adjacent nodes and receive the corresponding rewards. In addition, after moving to the next node, the agent could transit back to the previous state with a score of 0.

With this definition, the agent can be well-trained to attack graph types with several nodes and edges. However, with an attack graph with more nodes and a large enough number of training sessions, an agent can realize that moving consecutively between two nodes will yield a higher score.

In addition, with this definition, the author needs to mention precisely defining the action of the agent corresponding to specific states. The reason is that the attack graph generated by MulVAL is analyzed from the premises through the rules and then the resulting state.

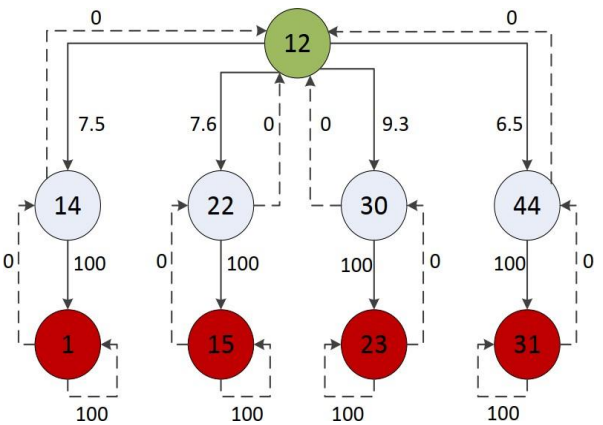


Figure 7. Environment for Automated penetration testing in the study [7]

This means that for the resulting state when returning to the pre-states.

The proposition will only be valid for the parent states that the agent already has. We consider an example in Figure 9. The example will be reviewed in the next chapter. When the agent moves to terminating node 1, the agent can only return to the previous state at node 20 or node 2, but not both.

To solve the above problems, instead of simply defining the state as the node's location, we define the state of the agent, including the node location and when the agent is at that node. Thus, when the agent returns to the visited node, it will be defined with the new state. Moreover, the agent will have points deducted if it repeats

the previous state instead of getting bonus rewards from these states. Doing this will cause the agent to try to reach the goal as quickly as possible instead of repeating previous states.

However, to solve the problem of allowing the agent to return to a previous location, it is crucial to define the agent's state separately when it reaches the next node from different states. This results in adding the previous nodes to the state's definition. However, the agent must be trained many times to try them all with this considerable number of states generated. Moreover, this takes much time, effort, and little practicality. Therefore, we will prevent the agent from moving back to the previous nodes if there is no directed link in the attack graph.

After taking action, the model can get a reward defined as follows:

- 100 if the state is the target.
- $\frac{\text{Base Score*Exploitation Score}}{10}$ if a vulnerability leads the state.
- 0.01 if the status is the attacker's location ("attacker-Located").
- 0.3 if the state is an intrusive node ("execCode").
- -1 if there is no connected node to the current node.
- -20 if the next state is the position of a previously transited node.
- 0.1 if it is not in the above cases.

With the example shown in Figure , we first determine the score of the security vulnerabilities, and the results are shown in Table 3.

For a real network, the agent will receive the corresponding score in the logical environment if the attack is successful but will receive a negative score if the attack is unsuccessful. Therefore, we need to define a reward function to calculate the reward for each state if the attack fails. The Reward Function for each failed attack on a device is defined as follows:

$$R(s, a) = \begin{cases} -20 * \text{failed time}, \\ \quad \text{if a failed} \\ \\ R(s, a) \text{ if a succeed} \end{cases} \quad (4)$$

In addition, when the attack fails, the episode will end, and the Q of the actions before the failed attack will be updated according to the following formula:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha \begin{bmatrix} r_t + \gamma * \max Q_{update}(s_{t+1},a_{t+1}) \\ -Q(s_t,a_t) \end{bmatrix} \quad (5)$$

In there:

- $Q(s_t,a_t)$ is the Q value of the state s_t, and the action a_t.
- r_t is the reward value of the state s_t.

TABLE 2. VULNERABILITY SCORES

Vulnerability	Base Score	Exploitability Score	Reward
CVE-2020-1938	9.8	3.9	17.6

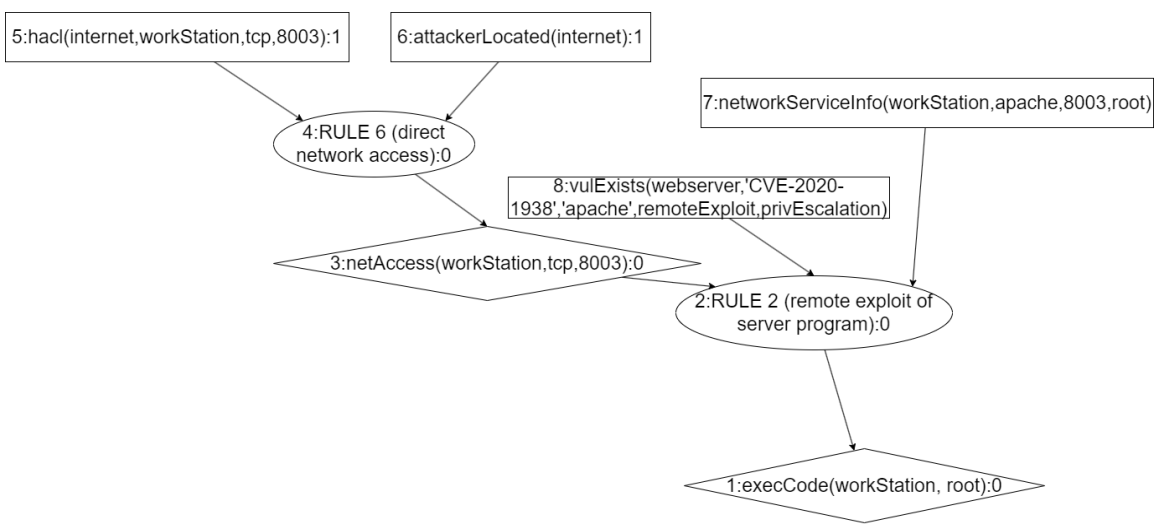


Figure 8. An example of the attack graph problem

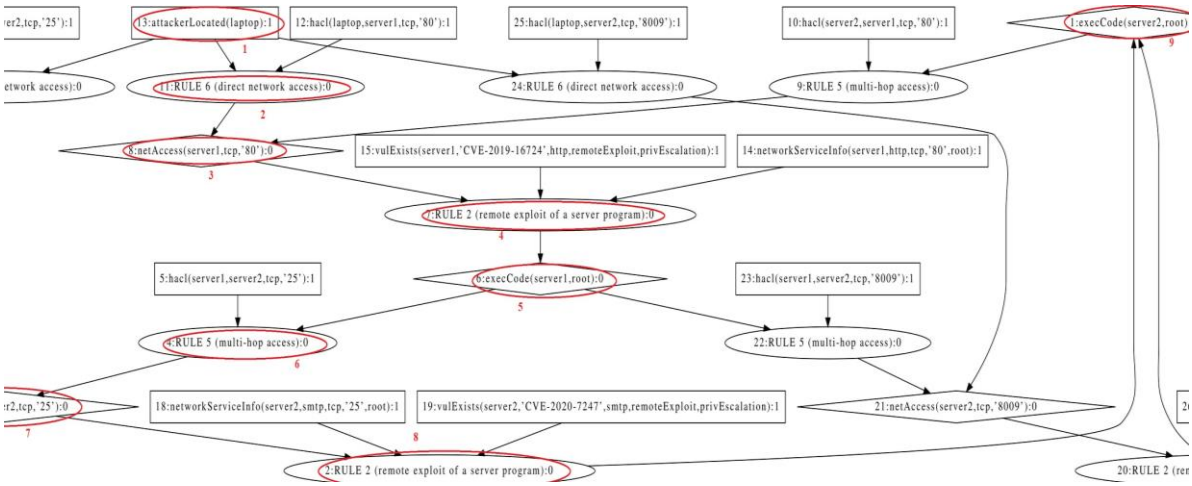


Figure 9. Label the attack graph with the depth-first search algorithm

TABLE 3. REWARD MATRIX

Node index	0	1	2	3	4	5	6	7
0	-1	-1	-1	-1	-1	-1	-1	-1
1	100	-1	-1	-1	-1	-1	-1	-1
2	-1	17.6	-1	-1	-1	-1	-1	-1
3	-1	-1	0.1	-1	-1	-1	-1	-1
4	-1	-1	-1	0.1	-1	-1	-1	-1
5	-1	-1	-1	0.01	-1	-1	-1	-1
6	-1	17.6	-1	0.1	-1	-1	-1	-1
7	-1	17.6	-1	0.1	-1	-1	-1	-1

- γ is the discount factor.
- $\max Q_{update}(s_{t+1},a_{t+1})$ is the maximum Q value after being updated.

V. HYPER-PARAMETER OPTIMIZATION AND REINFORCEMENT LEARNING MODEL EVALUATION

A. Automatic dataset generation method

To evaluate or optimize the model, we must have a labeled dataset. There are a few datasets that still need to be completed. Therefore, we propose to create datasets according to the following process:

1. Create an optimal attack path by attaching vulnerabilities corresponding to a predetermined optimal path to network model devices.
2. Generate other paths by adding random vulnerabilities, satisfying the condition that the total number of rewards created by the vulnerabilities is less than the total number of rewards the optimal attack path vulnerability could receive.
3. Create an attack graph with the MulVAL tool and reward matrix to assign weights to branches.
4. Use the depth-first search algorithm to find the path with the highest point as a label.

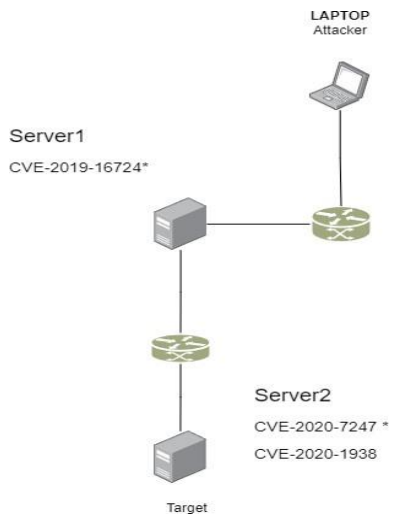


Figure 10. The process of creating network system data with labels

For example, in Figure 10, there are two servers in the network system, Server1 and Server2. We will create the optimal path from attacker to target through Server1 and Server2, including CVE-2019-16724 and CVE-2020-7247. Other paths are added and evaluated through the rewards that come from vulnerabilities. The non-optimal paths in the above case are attack paths using CVE-2020-1938.

Then, with the MulVAL tool, we create an attack graph for the network system and label

the attack path by the score of vulnerabilities. The optimal attack path is depicted in Figure 9. The winning nodes of the optimal attack path are marked in red and are listed in order from 1 to 9.

B. Hyper-parameter optimization for reinforcement learning model

Like other machine learning algorithms, reinforcement learning algorithms include parameters (hyper-parameters). These parameters play a significant role in the performance of the model. Specifically, with the Q-learning algorithm, these parameters include:

- Learning rate α (learning rate): the rate controls how fast the new value is updated compared to the old value.
- Discount factor γ: Balance ratio of immediate and future rewards. The larger the value, the more critical the future value is.
- Decay: coefficient to adjust the convergence speed of the algorithm.

To solve the exploration-exploitation dilemma in the reinforcement learning model, we model the parameter ε in the form:

$$\varepsilon=\varepsilon_{end}+(\varepsilon_{start}-\epsilon_{end})\times \frac{e^{-\text{step done}}}{\text{decay}\times\text{state num}\times 1000} \quad \text{with} \quad \begin{cases} \varepsilon_{start}=1 \\ \varepsilon_{end}=0.05 \end{cases} \quad (6)$$

In there:

- ε: allocation value for choosing the best action with random action selection.
- Decay: speed adjustment factor of ϵ.
- State num: the number of state nodes (only the first time) of the attack graph.
- ε_{start}: the starting value of ϵ.
- ε_{end}: the end value of ϵ.

In the above formula, we multiply the decay value by the number of states (first times only) of the network graph (state num) to balance input network systems of different complexity. With the number of moves in the environment equal to 0, ε has the value ε_{start}. However, as the number of moves increases gradually, the value of ε will decrease, and when step done $\rightarrow +\infty$ then $\varepsilon \rightarrow \varepsilon_{end}$. It means the greater the number of attempts of the agent in the environment, the less the agent will participate in the exploration process and more in the exploitation process.

As mentioned above, reinforcement learning aims to find the right policy to achieve the highest reward score the fastest. However, it is also necessary for the model to converge stably. Therefore, we must define the optimal function as having the highest final score and the highest mean score of the last 50 training episodes. So, the optimal function is defined as follows.

objective function=

$$-\text{average reward}-\text{scale}\times\text{optiomal reward} \quad (7)$$

Here,

- average reward: the average score of the last 50 training episodes.
- scale: coefficient to adjust the average and optimal score value achieved.
- optimal reward: the optimal value of the final reward.

In the function defined above, we add a scale value for testing with values that evaluate the importance of the optimal score (optimal reward) compared to the average score (average reward).

In addition, we need to limit the values the parameters can receive. The parameters are expressly limited as follows:

$$0< \varepsilon \leq 1$$
$$0< \gamma \leq 1$$
$$0< \text{scale} \leq 5$$
$$0< \text{decay} \leq 1$$

Finally, we summarize the limits of the parameters, as shown in Table 4.

TABLE 4. PARAMETER DOMAINS

Parameter	Minimum Value	Maximum Value	Data type
α	0 + △	1 - △	float
γ	0 + △	1 - △	float
scale	1 + △	5 - △	float
decay	0 + △	1 - △	float

TABLE 5. THE RESULT OF PARAMETERS

Scale	α	γ	Decay
0.0	0.5544	0.1189	0.2278
0.2632	0.4456	0.7722	0.01
0.5263	0.4456	0.7722	0.01
0.7895	0.4456	0.7722	0.01
. . .	. . .	. . .	. . .
4.7368	0.4456	0.7722	0.01
5.0	0.4456	0.7722	0.01

VI. EVALUATION RESULT

A. Hyper-parameter optimization result

The input data is generated through the data generation method with labels to conduct the training. The training data consists of 500 network systems with 500 attack path labels. In addition, the number of vulnerabilities on the attack path usually has 2-3 vulnerabilities, so the generated network model data is randomly labeled with 2-3 vulnerabilities on its attack path. We divide the domains of α, γ, and Decay into ten intervals with each parameter, and the domain of scale is divided into 20 intervals. After conducting many tests, the model finds the best value for the input data, as shown in Table 5.

According to the results in Table , the values of the tuples (α, γ, Decay) are the same as the scale values from the second value. The scale value of 0 will not be considered because the weight of the optimal point is removed and the value is not in the domain.

B. Evaluation Results

1. Evaluation with an auto-generated dataset with labels

We work with the automatically generated network model dataset and the real environment network model to evaluate the model. To evaluate objectively, we use the same method of creating datasets in the study [14] and evaluate the results between the two models. The generated network system dataset consists of 200 network systems, including different vulnerabilities labeled as optimal attack paths created with attack path depths ranging from 2 to 5 vulnerabilities. We also use the depth-first search algorithm to find the optimal path for the data for labeling. Because in AutoPentestDRL's model, the author is not interested in finding the shortest path but only in finding the path that gives the highest number of points back. Therefore, to compare the two models, we use networks with a persistent connection from the attacker's machine to the target machine. Therefore, the model will focus on assessing vulnerabilities in devices that attackers can attack.

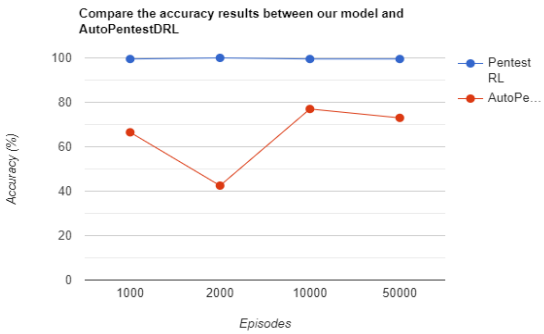


Figure 11. The model's accuracy compared to the model used in the AutoPentestDRL framework

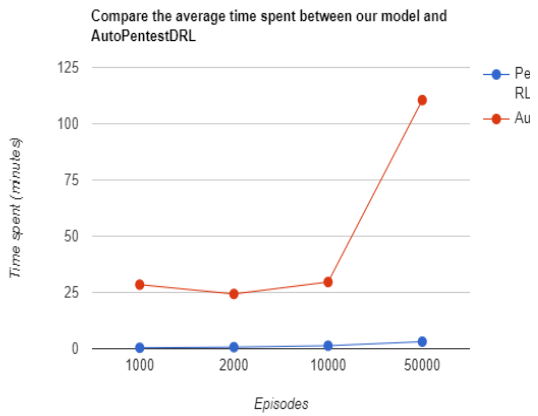


Figure 12. Compare the model's time spent against the model used in the AutoPentestDRL framework

TABLE 6. DIFFERENCES BETWEEN DATASET 1 AND DATASET 2

	Dataset 1	Dataset 2
Number of network models	200	200
Number of devices on attack paths	2-5	5-7
Number of vulnerabilities on each device	2-5	5-10

The evaluation results in Figures 11 and 12 show that our method has higher accuracy with the found parameter values than the model used in the AutoPentestDRL framework, with 100% accuracy when iterating with 1000 episodes and 99.5% with the number of episodes equal to

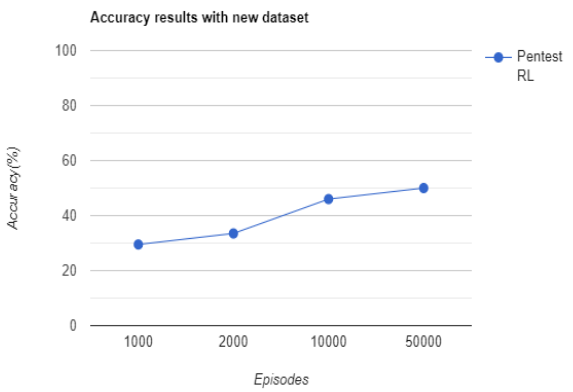


Figure 13. Model accuracy with the new dataset

500, 2000 and 5000 episodes. The time spent on our model is also much lower than that used in the AutoPentestDRL framework.

Our model works well with data sets on the same number of devices and vulnerabilities. present on networks. However, we create a second dataset following the same method to test the model's generalizability, as shown in Table 6. The second dataset includes 200 network systems with the optimal number of devices on the attack path consisting of 5 to 7 devices. In addition, each device includes between 5 and 10 vulnerabilities. After testing with the second dataset, the model returned the following results:

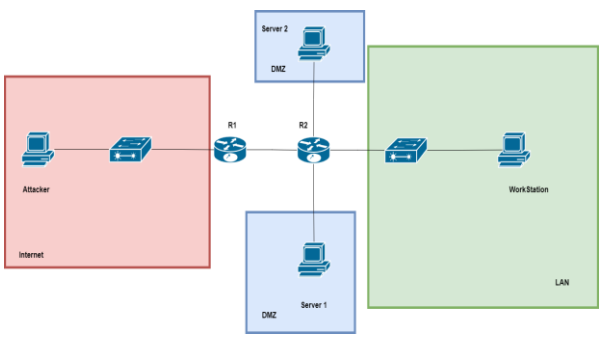


Figure 14. The real network topology in the experimeable

Figure 13 shows that our model can still analyze the attack graph and give accurate results up to 50% with 50,000 episodes.

2. Evaluation in the real environment

We need a network environment with real machines to evaluate the real environment. For convenience during testing, we use the GNS3 virtualization tool [15]. This tool allows us to create a network environment based on virtual machines such as VMware and VirtualBox. The system built on GNS3 interacts completely like the actual physical network system. In addition to automating the attack process, we use the Metasploit tool [16]. This tool allows us to use built-in attack modules to attack through one or more devices in the network [17]. In addition, this tool also allows control through the Python programming language [18].

The test network model is built as shown in Figure 14. The network system is divided into three areas: LAN, DMZ, and WAN, connected through network wires and switch and router systems.

We create vulnerabilities on devices similar to those in AutoPentestDRL's assessment scenario in the network for testing, including:

- Server1: Vulnerability CVE-2007-2447 on port 445.
- Server2: Vulnerabilities CVE-2004-2687 on port 3632, CVE- 2012-1823 on port 80, and CVE-2007-2447 on port 445.
- WorkStation: Vulnerability CVE-2020-24186 on port 80.

TABLE 7. THE SCORE OF VULNERABILITIES IN THE EXPERIMENT

Vulnerability	Base Score	Exploitability Score	Reward
CVE-2004-2687	8.6	9.3	7.998
CVE-2012-1823	10.0	7.5	7.5
CVE-2007-2447	6.8	6.0	4.08
CVE-2020-24186	3.9	10.0	3.9

Scores of vulnerabilities are shown in Tabale 8. Besides the real network model, as shown in Figure , router R2 is set up with ACL (Access Control List) to prevent direct access from the

WAN area to the LAN area. Thus, to be able to attack the target on the LAN, the attacker is required to attack through the servers located in the DMZ area.

We experiment with two scenarios to test the model's automatic attack capability.

- Scenario 1: An attacker attacks in a stable network environment.
- Scenario 2: An attacker attacks in the network environment where one of the two servers is isolated and is being upgraded.

Scenario 1: An attacker attacks in a stable network environment.

Experimenting and comparing with the model in the AutoPentestDRL framework, we obtained the logical, optimal path results in Table 8.

Based on Table 7, we see that with the AutoPentest-DRL model, the agent attacks the vulnerabilities in the order of Server1 with vulnerability CVE- 2007-2447, Server2 with vulnerability CVE-CVE-2004-2687 and then attacks workStation with vulnerability CVE-2020-24186. With our model, the agent attacks Server2's vulnerabilities with vulnerability CVE-2004-2687 and then attacks workStation with vulnerability CVE-2020-24186. Thus, the agent chooses a shorter and more optimal path with our model than with the other two methods. With the method used in AutoPentestDRL, the agent only considers attacking multiple vulnerabilities to get as many points as possible without considering whether the path is faster among the attack paths. However, an actor can attack vulnerabilities in a stable network environment in any order. As a result, all of the models' attack processes are successful.

TABLE 8. COMPARE THE LOGICAL OPTIMAL PATH AND ATTACKING RESULT WITH THE ALGORITHM MODEL USED IN AUTOPENTESTDRL

Model	Logical Optimal Attack Path	Success
AutoPentestDRL	20→51→8→7→6→14→13→12→11→55→3→2→1	✓
Our model	20→18→13→12→11→55→3→2→1	✓

Scenario 2: An attacker attacks in the network environment where one of the two servers is isolated and is being upgraded.

TABLE 9. THE RESULTS OF ATTACKS IN THE DYNAMIC NETWORK ENVIRONMENT

Attack Attempt	Logical Optimal Attack Path	Success
1	20→18→13→12→11→55→3→2→1	✗
2	20→18→13→25→11→55→3→2→1	✗
3	20→18→13→23→11→55→3→2→1	✗
4	20→51→8→7→6→4→3→2→1	✓

In this scenario, we will test our model's ability to choose the optimal path in a dynamic network environment. When the agent attacks, server2 is isolated from the network and is being upgraded. The results in Table 9 show that, after the attack fails on the optimal path in the logical environment, the agent redirects the attack and succeeds with other vulnerabilities in the real environment through server1 and workStation. Our model can choose the optimal path in the changing network environment.

In the AutoPentestDRL model, the agent loses its ability to update the states in the dynamic environment because the paths have been calculated and replaced in the simplified matrix. Therefore, the agent keeps the optimal path attack in the logical environment. Finally, the agent fails because the connection to server2 has been lost.

VII. CONCLUSION

In this article, we have proposed a method to build an environment and apply reinforcement learning algorithms to automated penetration testing. The model is optimized using a labeled dataset and evaluated in a real test environment. The system is integrated with intrusion detection and attack tools to test the existing system. Since then, the model's performance has been improved and evaluated to be appropriately applied to testing systems of different sizes. It is perfectly suitable for reducing the cost of penetration testing and maintaining regular guaranteed system testing. However, the

system is still limited in the network scanning process. It is required to find an approach from the scanner to all machines in the computer network. In addition, the system mainly depends on the results from the integrated tools, but the current number of system tools is small, and it is challenging to meet the needs of actual testing attacks.

In the future, we plan to focus on two directions. Firstly, we plan to integrate the attack automated test system with defense and log systems to build an automated network topology based on the data about network devices. Second, we plan to integrate other scanning and attack tools to detect a more significant number of vulnerabilities in the system and diversify the attack potential. Otherwise, we would also continue to experiment with other vulnerability databases to verify the adaptability of our model.

REFERENCES

[1] X. Y. B. T. B. C. M. J. Aileen G. Bacudio, "An Overview of Penetration Testing," International Journal of Network Security And Its Applications (IJNSA), pp. Vol.3, No.6, November 2011.

[2] M. E. Farmeena Khan, "A Comparative Study of White Box, Black Box and Grey Box Testing Techniques," in International Journal of Advanced Computer Science and Applications, June 2012.

[3] C. B. S. a. S. S. Vivek Shandilya, "Use of Attack Graphs in Security Systems," Journal of Computer Networks and Communications, 2014.

[4] R. L. a. K. Ingols, "An Annotated Review of Past Papers on Attack Graphs," in Lincoln Laboratory MASSACHUSETTS INSTITUTE OF TECHNOLOGY, 2005.

[5] L. P. S. Cynthia Phillips, "A Graph-Based System for Network Vulnerability Analysis," in Proceedings of the 1998 workshop on New security paradigms, 1998.

[6] J. H. S. J. e. a. O. Sheyner, "Automated generation and analysis of attack graphs," in (IJACSA) Proceedings of the IEEE Symposium on Security and Privacy, Berkeley, CA USA, May 2022.

[7] N. M. Y. Z. a. H. T. Mehdi Yousefi, "A Reinforcement Learning Approach for Attack Graph Analysis," in IEEE International Conference on Trust Security and Privacy in Computing and Communications (TrustCom), 2018.

[8] P. Đ. K. N. Đ. V. N. V. H. Nguyễn Mạnh Thiên, "Phát triển Framework ứng dụng AI hỗ trợ tự động khai thác lỗ hổng bảo mật," Journal of Science and Technology on Information Security, vol. 1, no. 13, pp. 80-92, 2022.

[9] "OpenVas," [Online]. Available: https://openvas.org. [Accessed 20 04 2022].

[10] S. G. A. W. A. Xinming Ou, "MulVAL: A Logic-based Network Security Analyzer," in USENIX Security Sympo, 31 July 2005.

[11] "Datalog," [Online]. Available: 2022]. Available: https://en.wikipedia.org/wiki/Datalog. [Accessed 20 6 2022].

[12] S. G. A. A. Xinming Ou, "MulVAL: A Logic-based Network Security Analyzer," in USENIX Security Symposium, 31 July 2005.

[13] B. J. K. W. a. F. D. Marcin Szpyrka, "Telecommunications Networks Risk Assessment with Bayesian Networks," in Computer Information Systems and Industrial Management Proceedings of the 12th IFIP TC8 International Conference CISIM, 2013.

[14] R. B. Y. T. Zhenguo Hu, "Automated Penetration Testing Using Deep Reinforcement Learning," in 2020 IEEE European Symposium on Security and Privacy Workshops (EuroS and PW), 2020.

[15] "GNS3," [Online]. Available: https://www.gns3.com. [Accessed 20 06 2022].

[16] "Metasploit," [Online]. Available: https://www.offensive security.com/metasploit-unleashed/pivoting/. [Accessed 20 06 2022].

[17] "Metasploit Pivoting," [Online]. Available: https://www.offensivesecurity.com/metasploit unleashed/pivoting/. [Accessed 22 06 2022].

[18] "Pymetasploit," [Online]. Available: https://github.com/DanMcInerney/pymetasploit3[Accessed 20 06 2022].

[19] S. W. Y. C. R. Z. a. C. W. Ianping Zeng, "Survey of Attack Graph Analysis Methods from the Perspective of Data and Knowledge Processing," Security and Communication Networks, vol. 2019, no. 2031063, 2019.

ABOUT THE AUTHORS

Nguyen Thanh Cong

Workplace: Faculty of information technology, Le Quy Don Technical University.

Email: nguyencongg1@gmail.com

Education: Student in Information System Security specification, Le Quy Don Technical University.

Recent research interests: Information security; Penetration testing.

Nguyen Viet Hung

Workplace: Faculty of information technology, Le Quy Don Technical University.

Email: hungn@mta.edu.vn

Education: He received his BSc, MSc and PhD degrees in Computer Science from Moscow Institute of Physics and Technology in 2006, 2008 and 2012 respectively.

Recent research interests: Information security; Penetration testing; Intrusion detection.