Master's Thesis

Automated Penetration Testing Using Deep Reinforcement Learning

HU Zhenguo

Graduate School of Advanced Science and Technology
Japan Advanced Institute of Science and Technology
(Information Science)

March, 2021

Abstract

Penetration testing is a new network security technology developed in recent years, which has great practical application for computer network and security. It is a security testing and evaluation method that uses hacker techniques and methods to discover the security vulnerabilities of the target machine. By following this process, one can gain control of the system, access the confidential data, and discover the possible security risks that may affect the continued operation of the business. The ethical hackers, which are called pentesters, will conduct an in-depth security detection of the target network to find out the vulnerable links, and use various realistic attack methods to detect the possible vulnerabilities.

The biggest difference between penetration testing and hacker intrusion is that penetration testing is authorized by the customer. It uses controllable and non-destructive methods to find weaknesses in the target and network equipment, helps managers to know the problems located in their own network, and provides security suggestions to help improve the security of their system. However, currently penetration testing is performed mostly manually and relies mostly on the pentesters' experience, and there are no tools to intelligently analyze the network situation and discover the potential attack paths in a network system.

In order to solve this problem, many researchers try to find some methods that mimick the penetration testing process to discover the attack paths. One of the representative methods is called PDDL, which means Planning Domain Definition Language. It uses planning algorithms, such as graph planning and partial-order planning, to convert attack paths into PDDL expressions. But these methods do not perform well in the discovery of potential paths, and the degree of automation is not high. Another representative method is FF-Replan, which is a dynamic reprogramming algorithm that decomposes the probabilistic programming problem. The algorithm transforms the non-deterministic programming problem into a deterministic programming problem, then uses the FF (Fast Forward) programming algorithm to achieve automated attack path planning.

However, these methods have some shortages, as they need to delete non-deterministic information for planning, and have difficulties in dealing with path finding problems with multiple uncertain attack paths. For this reason, we present another method by which we reinforcement learning techniques into the field of cybersecurity, and use the powerful analysis capabilities of the neural networks to automatically plan the attack paths. Since reinforcement

learning does not require data to have precise labels, it is very suitable for attack path analysis.

In this thesis, we propose an automated penetration testing framework named AutoPentest-DRL, which we designed and implemented to address the shortcomings mentioned so far. The key idea is to employ deep reinforcement learning (DRL) to plan the attack path, and employ other penetration tools to automate the process of penetration testing. To realize this goal, we built a DQN Decision Engine to select the correct attack path according to network and vulnerability information. The input for the decision engine is the matrix representation of the attack tree, and the output is the most feasible attack path. We also employ a topology generator to create enough network topologies in order to increase the model's adaptability. Furthermore, the Depth-First Search (DFS) algorithm is used to simplify the input matrix. In this way, the AutoPentest-DRL can automatically give the corresponding attack path according to the input network information.

In order to make it possible to conduct penetration testing in a real network environment, we adopted Nmap to retrieve the necessary information by scanning the real network environment. The information of vulnerabilities is extracted from the scanning report of Nmap, and is combined with the network topology data in order to send them to the DQN Decision Engine. Penetration tools such as Metasploit have also been integrated in order to to execute automated attack commands.

We first discuss the basic architecture of this framework and the implementation of AutoPentest-DRL. Then, we demonstrate the efficiency of this framework by evaluating it both in logical and real network environments. In logical networks, the average accuracy of the two different topologies we studied is 0.932. In real networks, the framework can employ three different vulnerabilities to get the root privilege of the corresponding serves, and finally, it can copy the test Trojan to the target machine successfully. Our results show that AutoPentest-DRL is suited well for both environments and it is possible to apply it to real systems.

In the future, we plan to do more research on fast attack path discovery techniques in large-scale network scenarios to support penetration testing on complex networks. We also plan to incorporate the generated feedback information during the process of penetration testing into the attack path discovery algorithm to dynamically adjust the attack path. Finally, we will improve the compatibility of our system by integrating other penetration testing tools, such as Nessus and Cobalt Strike. We really hope that our research can promote the development of automated penetration testing, and also inspire other researchers.

Contents

List of Figures

List of Tables

Acknowledgement

At first, I would like to express my sincere appreciation for my supervisor, Associate Professor Razvan BEURAN. Thanks so much for his advice and his instructive comments on the direction of my own research. During the process of writing the thesis, he gave so much helpful guidance on the difficulties and doubts that I faced, and invested a lot of effort and energy. His kind assistance helped a lot with my research in JAIST.

At the same time, I would like to sincerely thank my second supervisor, Professor Yasuo TAN for his patient instructions and support. His guidance and constant encouragement help me to continue my study. I would also like to thank Associate Professor Shinobu HASEGAWA. With his careful guidance and advice, I finished my minor research very well.

In addition, I would also like to thank my friends in our lab. From their helpful support and suggestions, I got great inspiration for my thesis. I am also grateful to the authors in the reference literature. After reading their research, I understand how I can start my research topic.

Finally, I would like to thank the thesis review teachers for their hard work. I am sincerely grateful to my family and friends. Thanks to their good support and encouragement, I was able to complete my master's thesis successfully.

Chapter 1

Introduction

In recent years, with the rapid expansion of network connections and the frequent occurrence of security incidents, especially the large-scale opening of the Internet and the access of financial networks, cybersecurity has become an increasingly prominent problem, more and more systems are threatened by intrusion attacks. In 2014, Yahoo suffered serious network attacks, threatening the accounts of nearly 1 billion users, which is the biggest vulnerability found in the history of cybersecurity. In September 2018, Facebook officially admitted that due to a token access vulnerability, hackers could take over 50 million user accounts, and about 90 million users were affected. One of the Japanese biggest cryptocurrency exchanges *Bitpoint* was attacked in July 2019, which caused the loss of 32 million dollars.

In order to address this issue, an effective way is to utilize penetration testing to test the security of the company's or government's network environment. Penetration testing is an authorized attack process that is always been used to evaluate the system's network security, especially on its network security weaknesses and shortcomings through executing ethical attacks. The managers can see the potential threats for malicious attackers to enter the company or cause damage to the company's data assets by simulating real-time attacks [1]. This kind of approach is been built, and several commercial programs are available to help the pentesters (white-hat hackers) in executing the tedious jobs [2]. In summary, penetration testing plays a significant role in cybersecurity.

The process of penetration testing is currently done mostly manually, as penetration testers need to attack the test system and use verification attacks to test different network environments by themselves. McDermott divides the penetration testing process into six stages: define penetration testing objectives, conduct background studying, create imaginary flaws, verify surmise, generalize discovered flaws and remove discovered flaws [3]. Because the basic knowledge of network security is not easy to formalize, and prone to human errors, so this process is an extremely laborious, time-consuming, and complex task [4]. Thus, with the development of technology, more and more people try to model the target environment and generate attack plans through attacks based on different models. The Core security company has adopted this penetration test method since 2010, and adopted a variant of metric FF system as the attack plan in the research which is called 'Core IMPACT' [5].

With the vigorous development of computer technology, more and more researchers are migrating their research in the field of chess to penetration testing, and automated penetration testing is gaining popularity. Through automated analysis, automated penetration testing can discover the potential vulnerabilities of the target network and host, and invoke the attack payload to verify those kinds of vulnerabilities [6, 7]. The establishment of the attack tree is a very important part of ensuring automated penetration testing. It is first proposed by Schneier in 1999, the idea is to model security danger on a specific network environment [8].

At the same time, Obes tried to employ plan domain definition language (PDDL) to the field penetration testing in order to use path planning technology to find the attack paths [9]. Boddy first employed artificial intelligence in the process of penetration testing [10], which leads to the insertion of the cybersecurity area into the 2008 International Planning Competition. Several years later, the company of Core Security tried to use simulated external attacks to make a plan for penetration testing [11]. Meanwhile, Yousefi applied reinforcement learning (RL) to make an analysis of the attack tree [12], which used Q-learning to get the most possible attack path, but it still can not apply to the real network environment.

All these tools have improved the efficiency of penetration testing, but there are still some problems. From the host level, these tools simply integrate existing network attack tools, lack the ability to reason about network attack and defense knowledge, and cannot intelligently select attack loads and configure load parameters for penetration testing based on the status of the target host; from the network level, most of these penetration testing tools perform vulnerability assessments for a single host, and cannot assess the vulnerability of the entire target network, and also lack the ability to discover potential attack paths of the target network.

To get over this kind of problem, a promising approach is turning the discovery of attack paths problem into a description of a reinforcement learning problem, basically expressing penetration testing as an environment of reinforcement learning. Different from reinforcement learning, deep reinforcement learning combines reinforcement learning with deep learning, and uses a trial and error method to find the best solution, which is more suitable for attack path planning and analyzing attack tree.

In this paper we present the framework named AutoPentest-DRL that we built with the goal of executing penetration testing automatically for a real network environment. Our main contributions are:

- We employ a reinforcement learning (RL) method based on deep Q-learning (DQN) to select the optimal attack path of automated penetration testing. The policy prioritizes selecting the path which has the biggest reward.
- We randomly generate a great number of network topologies to improve the diversity of the training model. We also use the DFS algorithm to simplify the attack path matrix to make it more suitable as input for DQN.
- We integrate scanning tools such as Nmap into the framework to realize automatic detection of the target network, and employ the RPC API to communicate with Metasploit to execute attack commands.

The rest of the thesis is organized as follows. In Section 2, we provide an overview of the algorithms and techniques related to reinforcement learning and attack tree, the penetration tools that we plan to use and some related work that how the other researchers use attack path planning to solve the similar problems. Then in Section 3, we provides details of the proposed automated penetration testing framework architecture. Next we propose an experiment that we designed to make a evaluation of the framework both on a logical network and a real network environment in Section 4. Finally, we conclude the paper in Section 5 and propose our future developments.

Chapter 2

Background and Related Work

2.1 Reinforcement Learning

Deep learning has contributed to dramatic advances in the scalability and performance of machine learning over the past years [13]. One of the most famous applications is the sequential decision-making setting of reinforcement learning (RL). Reinforcement learning maps environment space and action space to each other, so that the agent can get the maximum cumulative reward in the process of training [14]. Delayed return and trial-and-error are the main characteristics of reinforcement learning. It is very important to choose different actions to get the maximum reward [15]. The final target of reinforcement learning is to find the best policy in which the agent will get the biggest reward, so the agent needs to try and try many times to understand how to communicate with the environment [14, 16].

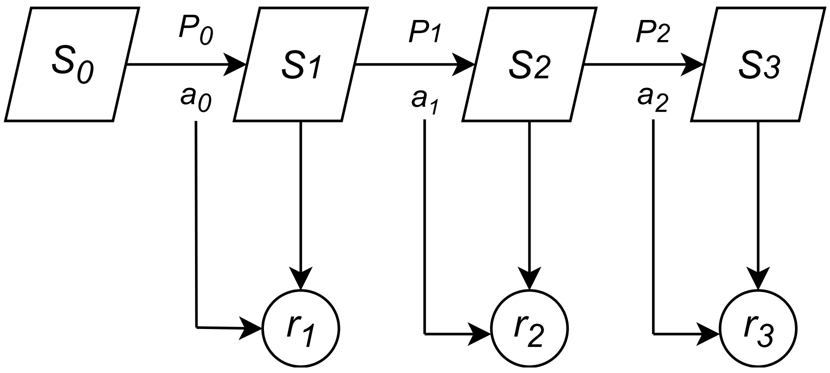


Figure 2.1: Overview of MDP architecture.

The process of reinforcement learning (RL) can be explained through the Markov Decision Processes (MDP). An MDP model can be separated into four different parts: S, A, P, R. S is the state space; A is the action space; P(s'| s, a) is the possibility that the agent adopts the action from s to s'; *R(s, a)* is the calculated reward. In Fig.2.1, an agent of MDP process is in the initial state S_0, after performing the action a_0, it transitions to the state S_1 according to the probability P_0, and uses the reward function to obtain the reward value r_1 according to a_0 and S_1. When the system state transitions from the current state to the next state s', the function of reward can be denoted as r(s'| s, a).

So as to the reinforcement learning process, as Fig.2.2 shows, t is the time step. First through the environment, the agent will get the current state S_t. Then the agent will do the action A_t. Next the next state S_{t+1} and the reward R_t which are corresponding to the time step will be created by the environment.

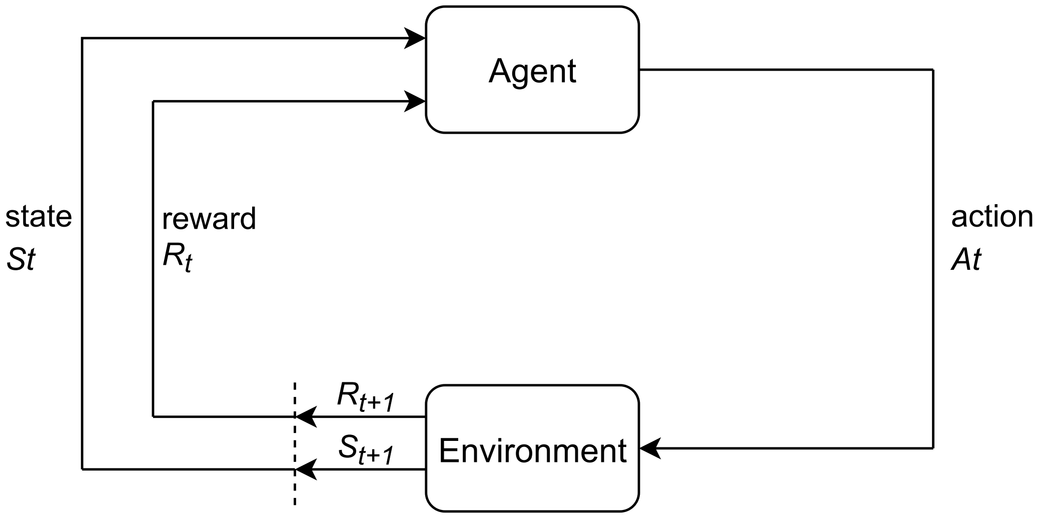


Figure 2.2: Overview of RL process.

In reinforcement learning, the objective is to maximize the cumulative rewards which can be used to choose the action. Strategy π: S→A is a mapping from state space to action space. It means that the agent chooses action a_t in the state s_t, executes it and transfers from current state to next state s_{t+1} with the possibility *f(s_t, a_t)*, and accepts the reward r_t from the environment feedback simultaneously. In the algorithm model, the reward and the discount factor γ need to be multiplied with each other, in this way a reward function is been defined to calculate the corresponding reward at the different time t:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \tag{2.1}$$

Because if it is located far away from the current state, the reward value $\gamma \in [0, 1]$ will get the great uncertainty The discount factor (γ) is always been used to calculate for the model's uncertainty.

For that reason, the value function is been imported to as the representative of the "value" for a state, for instance, for the specific state, the following formula is the expectation of the future rewards:

$$V^{\pi}(s) = E\left[\sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \mid s_t = s\right] \tag{2.2}$$

The function of state action is $Q^{\pi}(s, a)$, that has connection with the execution of action a which is located in s. After training many times until the agent finds the end target, the reward in this process can be described as the following formula:

$$Q^{\pi}(s, a) = E\left[\sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \mid s_t = s, a_t = a\right] \tag{2.3}$$

Our goal is to find the best performing strategy. We think that if one of the strategies has the greatest reward, then it is the best performing strategy. We take it as the result of our final policy. The action state function of finding the strategy is shown as follows:

$$Q^{*}(s, a) = \underset{\pi}{max}\, E\left[\sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \mid s_t = s, a_t = a\right] \tag{2.4}$$

RL in automated penetration testing generates the best policy through its interactions in its environment without prior knowledge. It has successfully captured the complexity and uncertainty of penetration testing [17]. RL based model for penetration testing has been used in recent research works [18].

2.2 Deep Reinforcement Learning

The basic idea of deep reinforcement learning (DRL) is to combine the reinforcement learning and deep learning. It integrates deep learning's understanding of perception problems such as vision. At the same time, we also need to understand how it makes the choice.

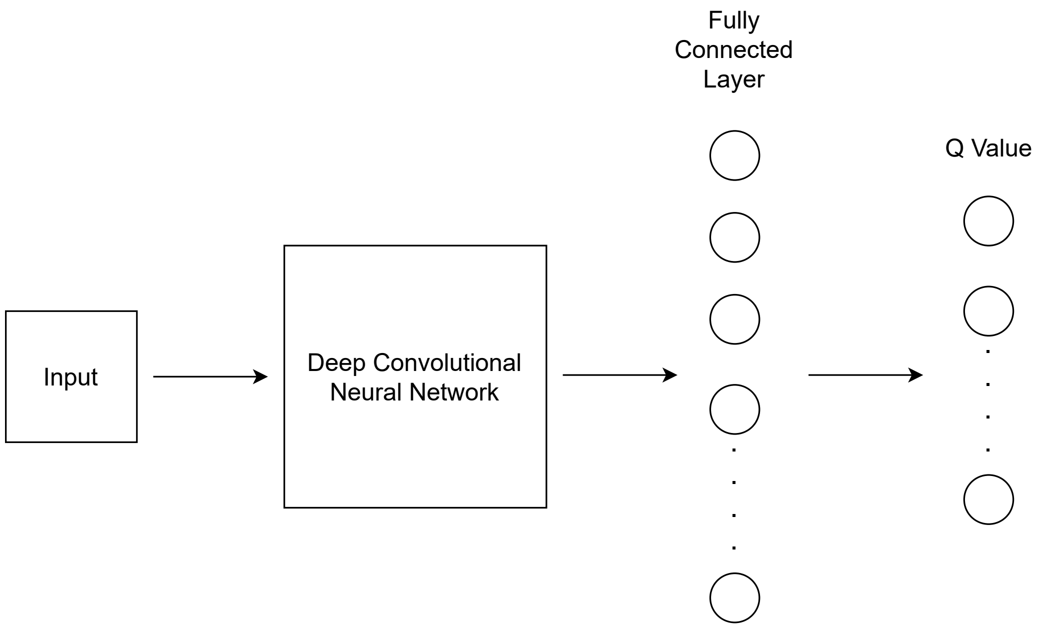


Figure 2.3: Overview of the architecture of DQN model.

In the deep reinforcement learning, the state space is been obtained by the "agent", and it chooses the suitable and corresponding actions. This process is related to the different policies. The state of the environment will be changed, and finally the model will gain the reward [19]. The emergence of DRL makes RL technology truly practical and can solve complex problems in real-world scenarios.

Deep reinforcement learning is been separated into 3 different functions: value-based function, policy-based function, model-based function [20]. The RL model has been used by many existing methods which utilize the Q-Learning approach [21]. One of the representative of value-based functions is Deep Q-Network (DQN), it is first proposed by Mnih et al. in 2013 which is been used to learn strategies directly from pixel images to play Atari games [22, 23], it combines the Q-Learning algorithm [24] with the Convolution Neural Network (CNN) in the inner of the reinforcement learning, so as to create the advanced DQN model.

A typical DQN model architecture is been provided in Fig. 2.3. DQN model contains both input and output. In this figure, it is assumed that the input is a picture, which is transformed non linearly by the convolution layer and the fully connected layer, and in the end it will output a q value.

Fig. 2.4 indicates a classic training process of the DQN model. The target values are constructed using a designated target network $Q(s, a; \theta^{-})$ (using

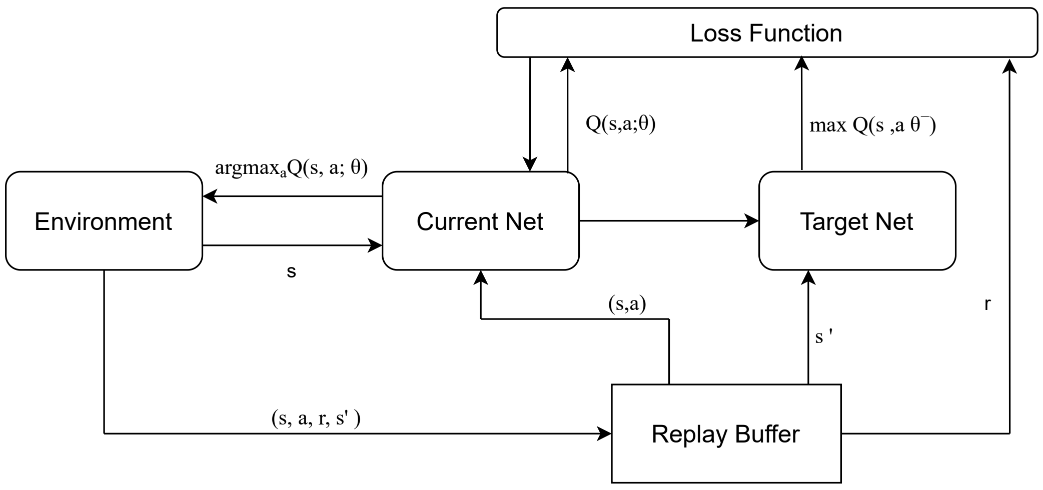


Figure 2.4: Overview of the training process for DQN.

the previous iteration parameters θ^-), where the expectation is to take 'w.r.t.' as the sample allocation of experience transitions in the replay buffer [25]. The DQN loss is minimized by using a Stochastic Gradient Descent (SGD) variant, sampling mini-batches from the replay buffer. Additionally, DQN model requires an exploration procedure to interact with the environment. The number of new experience transitions (s, a, r, s') added by exploration to the replay buffer. Fig. 2.5 is the basic architecture of DQN method. By using this kind method, we can utilize the DQN to find the optimal attack path.

In order to determine the parameters of 1 and the discount factor γ, and improve the generality and stability of the algorithm, we need to reduce the reward of deep Q-learning network as much as possible. At present, deep reinforcement learning is widely used in face recognition, automatic driving and other fields, which fully proves that deep Q-learning method has strong adaptability and versatility.

2.3 Attack Tree

2.3.1 Attack Tree Model

The attack tree gives us a systematic method to show the security of the system according to the different attacks. Basically, that path represents an attack on the system in a tree structure, the attack starts from the top node, the target is the root node, and the different way to achieve the target is the leaf node. Attack trees are a formal method of performing threat and risk

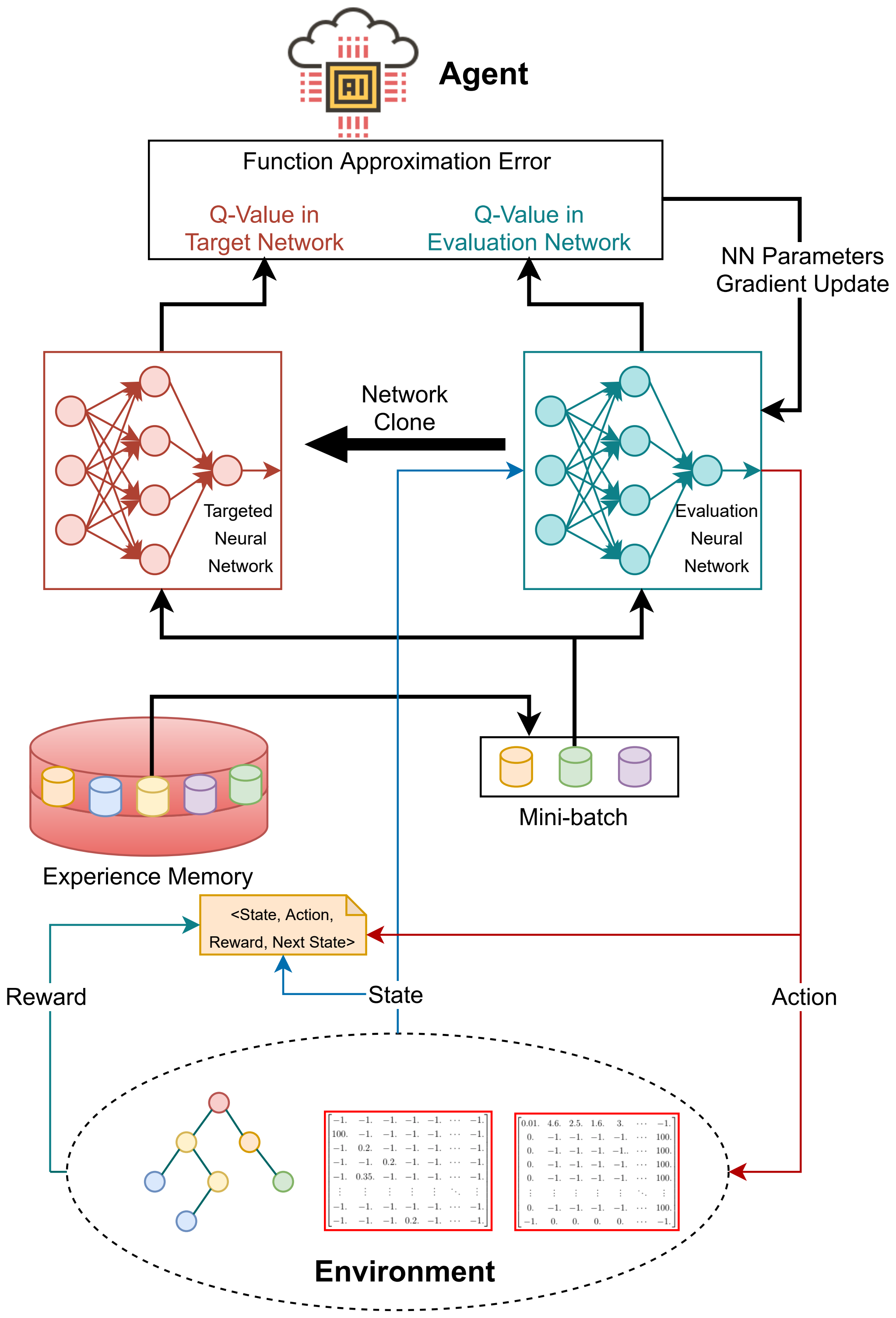

Agent
Function Approximation Error
Q-Value in Target Network
Q-Value in Evaluation Network
NN Parameters Gradient Update
Network Clone
Targeted Neural Network
Evaluation Neural Network
Mini-batch
Experience Memory
<State, Action, Reward, Next State>
Reward
State
Action
Environment

Figure 2.5: Architecture of Deep Q-learning.

analysis on a defined system.

The attack tree model was first proposed by Schneier in 1999 [8]. The model uses a kind of tree representation of the interdependence between attack behaviors and attack steps in order to analyze the security threats of the system. In the inner of the attack tree, every node is a kind of action or target, the root node located at the bottom of the tree indicates that the attacker wants to reach the final goal. The leaf node at the end of the back tree is the specific attack method. There are some advantages of this analysis method that it has a clear structure, combines with the nature of network attacks, and the data representation of the attack tree is more flexible and has the characteristics of reusability [26].

By assigning values to each node, we can do some basic calculations based on this tree structure to describe various attack methods against the overall target, so it is still feasible to apply it to the field of attack. According to a given attack tree, we can start from a certain leaf node of the tree to find a path that can achieve our attack goal and has a relatively low cost.

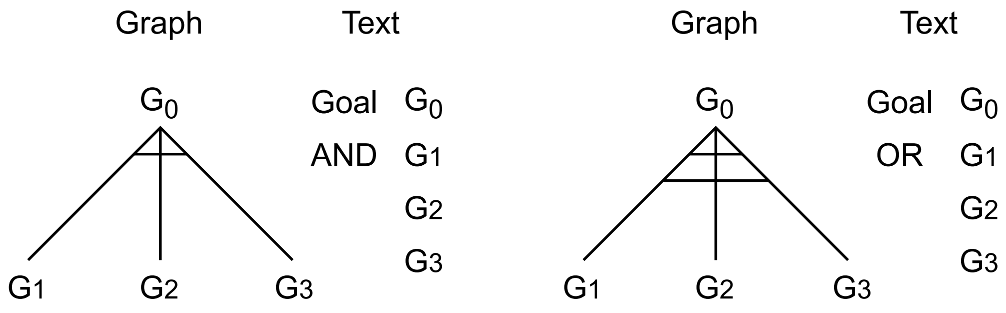


Figure 2.6: Attack tree node representation.

The structure of the attack tree model is clear and intuitive, which fits the nature of network attacks, and is reusable and easy to expand. It is often used for risk analysis and designing countermeasures against attacks.

2.3.2 MulVAL

The meaning of MulVAL is “Multi-host, Multi-stage Vulnerability Analysis Language”. It is an open-source security practitioner tool, the system administrators cause it to generate the real attack tree for a network environment. At the same time, it can be used to help enterprises manage the use of different networks, so as to reduce the security risk of enterprises [27]. By using the attack graph automatic generation technology of the MulVAL tool, the attack graph does consider all the vulnerabilities that can be used by the attacker, and makes the method of network security assessment move from manual to automated, and evaluate it in a more objective and precise manner. It helps further improve the handling of different vulnerabilities.

Based on the different network topology properties, the complexity is also different, it is between $\mathcal{O}(n^2)$ and $\mathcal{O}(n^3)$. Because of this, other automation tools can use MulVAL to improve the management efficiency of the company. So many small and medium-sized companies are willing to use it.

2.4 Penetration Testing Tools

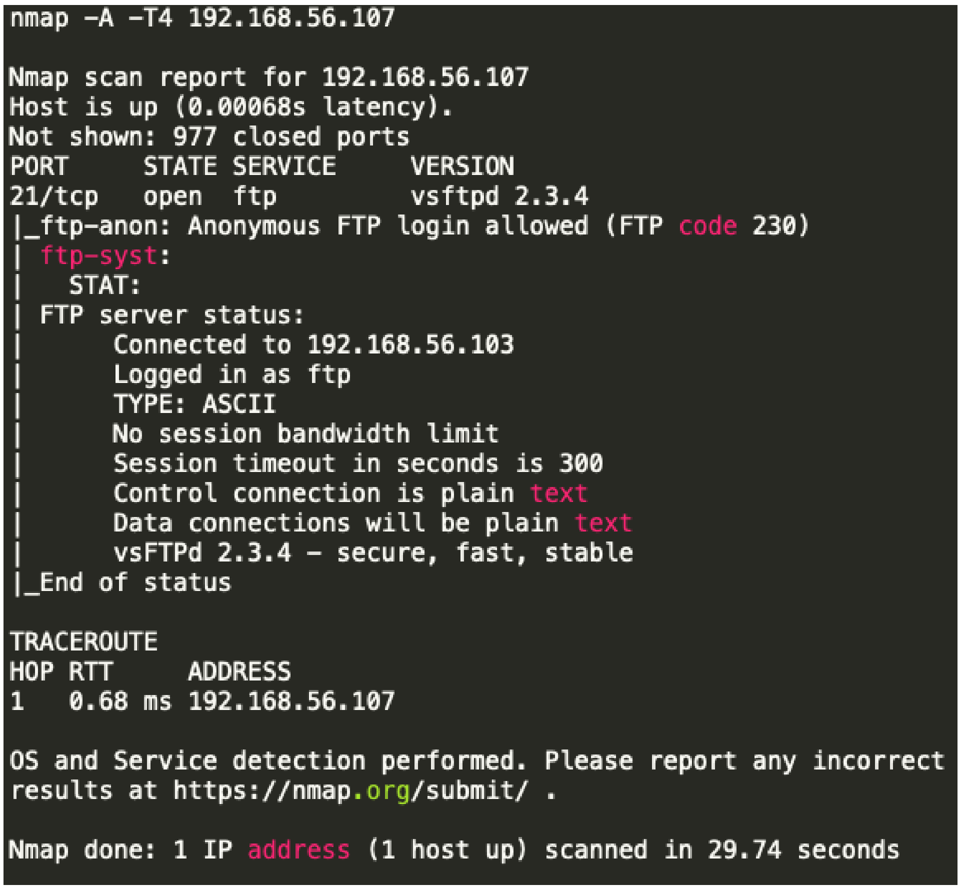
2.4.1 Shodan

Although people currently think that Google is the most powerful search engine, Shodan is the most terrifying search engine on the Internet. Unlike Google, Shodan does not search for URLs on the Internet, but directly enters the back channel of the Internet. Shodan can be said to be a “dark” Google, constantly looking for all the servers, cameras, printers, routers, etc. connected to the Internet. Google cannot search for device information on the Internet, while Shodan can obtain device information. And many times servers, cameras, printers, routers and so on. require an administrator password to log in. Shodan is used to search for online devices in the cyberspace. You can search for specific devices or specific types of devices through Shodan. The most popular searches on Shodan are: *webcam*, *linksys*, *cisco*, *netgear*, *SCADA* and so on.

The working principle of Shodan is to scan the entire network of devices and capture and analyze the banner information returned by each device. By understanding this information, Shodan can know which Web server is the most popular during the network, or how many exist in the current network. Fig. 2.7 provides an example result of the Shodan search engine.

```
"_shodan": {
        "crawler": "d264629436af1b777b3b513ca6ed1404d7395d80",
        "id": "970ccb1e-05d9-483a-a1fc-13dfa07d419b",
        "module": "https-simple-new",
        "options": {},
        "ptr": true
    },
    "asn": "AS4812",
    "cpe": [
        "cpe:/a:apache:http_server:2.2.15"
    ],
    "hash": -1963466945,
    "info": "(CentOS)",
    "os": null,
    "port": 8081,
    "product": "Apache httpd",
    "timestamp": "2019-11-26T04:07:02.615692",
    "transport": "tcp",
    "version": "2.2.15",
    "vulns": {
        "CVE-2010-1452": {
            "cvss": "5.0",
            "verified": false
        },
        "CVE-2011-0419": {
            "cvss": 4.3,
            "verified": false
        },
        "CVE-2011-3192": {
            "cvss": "7.8",
            "verified": false
        },
        "CVE-2011-3348": {
            "cvss": 4.3,
            "verified": false
        }
    }
```

Figure 2.7: Example of Shodan searching result.



```
nmap -A -T4 192.168.56.107

Nmap scan report for 192.168.56.107
Host is up (0.00068s latency).
Not shown: 977 closed ports
PORT     STATE SERVICE     VERSION
21/tcp   open  ftp         vsftpd 2.3.4
|_ftp-anon: Anonymous FTP login allowed (FTP code 230)
| ftp-syst:
|   STAT:
| FTP server status:
|      Connected to 192.168.56.103
|      Logged in as ftp
|      TYPE: ASCII
|      No session bandwidth limit
|      Session timeout in seconds is 300
|      Control connection is plain text
|      Data connections will be plain text
|      vsFTPd 2.3.4 - secure, fast, stable
|_End of status

TRACEROUTE
HOP RTT     ADDRESS
1   0.68 ms 192.168.56.107

OS and Service detection performed. Please report any incorrect
results at https://nmap.org/submit/ .

Nmap done: 1 IP address (1 host up) scanned in 29.74 seconds
```

Figure 2.8: Nmap scan report.

2.4.2 Nmap

Nmap ("Network Mapper") is a free utility to scan the network environment. It is the world's best port scanner and a prominent piece of our host security instruments [28]. It will find the basic information of the target machine by using IP address to or host name. It can also find the vulnerability information which includes CVE id, running port and so on. Nmap has many different operating parameters. Through different combinations, we can find various network information of the target network, including different ports, product versions, and possible vulnerabilities [29].

A typical Nmap scan is shown in Fig. 2.8. Argument $-A$ is been used to let version and operation system can be detected. Argument $-T4$ is been used to faster the execution.

2.4.3 Metasploit

By using Metasploit, we can use different expositions to attack the tested host [30]. It basically contains the current commonly used vulnerability machine utilization methods, which is very suitable for white-hat hackers. It can run on Linux, Mac, Windows and other kinds of operating systems. The goal of Metasploit is to build a penetration testing tool that even novices can easily use to execute attack testing.

Metasploit is been built to simplify the lives of security experts. At first, the main users are the people who are working on cybersecurity and other people who are really interested in cybersecurity. Within the guidance, these people can try to do penetration testing, Shellcode writing and vulnerability finding. At the same time, Metasploit provides an extensible model to integrate load control (Payloads), encoders (Encoders), no-operation generators (Nops) and vulnerabilities. It integrates common overflow vulnerabilities and popular Shellcode on various system platforms, so Metasploit is an effective way to analyze some high-risk vulnerabilities [31].

The scope of the Metasploit payloads starts from normal command shells to powerful post-exploitation shells like Meterpreter. It is a kind of post-exploitation agents which employ the protocol to execute actions. [32]. Through the Meterpreter, the attacker can upload Trojan to the target machine, execute malicious programs, and interact with the target host. As a backdoor with extremely high permission, the Meterpreter can upgrade ordinary Shellcode to Meterpreter through the code 'sessions - u id'.

RPC service is a set of message types and remote attack methods, which can provide a way for external applications to interact with web services. Metasploit can establish an RPC server through HTTP based to interact with Metasploit command console or session. For example, it can start the main framework and use the vulnerability to penetrate other hosts. At the same time, it can also be combined with other penetration testing tools to achieve good performance.

2.5 Penetration Testing Using Attack Path Planning

Over the past decades, there are many kinds of different attempts about automated penetration testing. Schneier first presented an attack tree model to describe attack [8]. But the attack tree model has some limitations, it only faces the attack path of single target. In 2002, Sheyner proposed an attack graph model, the idae is that every environment of penetration testing is a sequence of the intruders' different actions [33]. Obes et al. tried to use plan domain definition language (PDDL) to bridge penetration testing and planning in order to find attack paths [9]. Fig. 2.9 is the attack path planning method which uses PDDL representation to plan the attack path. It first converted the attack path discovery into PDDL based on vulnerability and network scenario information representation; then used a deterministic planning algorithm to find the attack path. According to the different planning models, the algorithm of attack path planning has been divided into three types: based on planning diagram technology, based on partial-order planning, and based on hierarchical task network. Khan employed PDDL to implement vulnerability assessment automatically in 2017 [34]. And there is also a good product named "Core Impact" that applied PDDL on commercial products [11].

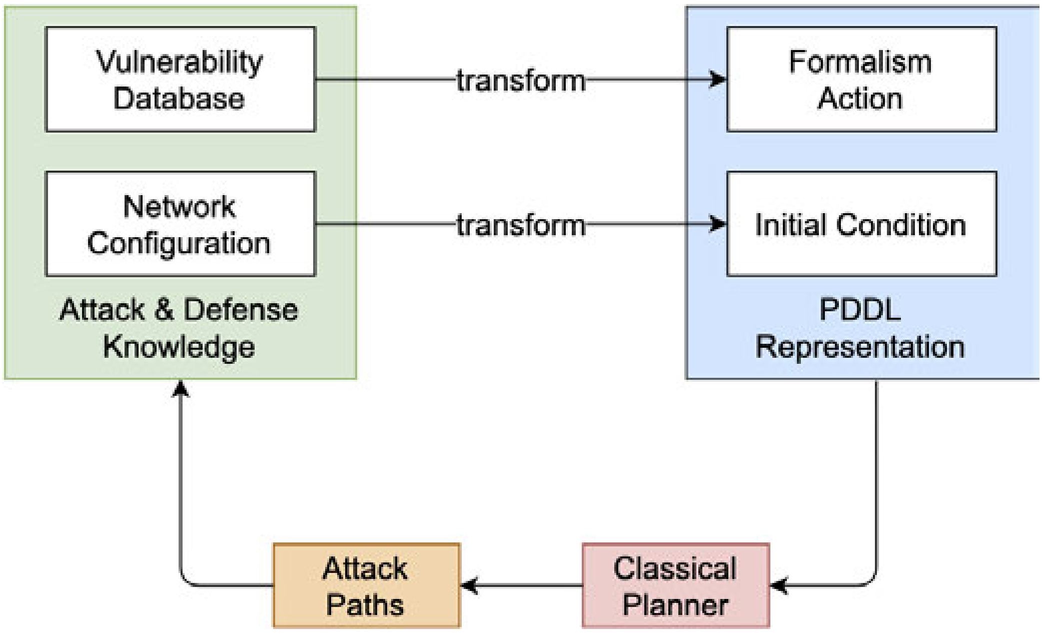


Figure 2.9: Attack path planning using PDDL.

There is also an approach named Hierarchical Task Network (HTN) which is based on the task and task decomposition. The advantage of this algorithm is that it can analyze planning problems from a high-level abstract perspective, and it can ignore the specific implementation details of the bottom layer simultaneously, which is easy to understand. The disadvantage is that it is often necessary to customize specific problem decomposition methods and implementation methods to deal with specific problems. There is another similar method named HTNLearn which converts the constraint problem of the task decomposition method into a CSP problem and uses a weighted MAX-SAT solver to solve it.

However, these methods mainly solve the problem under static, deterministic, and fully observable conditions, but penetration testing is a kind of dynamic, non-deterministic, and under partial observation conditions process, all those kinds of methods need enough information about a specific network. Therefore, the study of non-deterministic planning technology has great significance in realizing attack path planning. One of the representative methods is attack path planning based on Determinizing technology. It transforms non-deterministic problems into multiple deterministic planning problems, then uses deterministic planning algorithms to solve the transformed problems. Another representative approach is FF-Replan which is based on probability optimization technology. The algorithm first randomly selects a certain behavior result from a variety of possible behavior results. According to this, it transforms the non-deterministic programming problem into a deterministic programming problem, and then uses the FF (Fast Forward) algorithm to achieve attack path planning. If an unexpected state is encountered, the above process is iteratively performed with this state as the initial state.

Chapter 3

Framework Architecture

3.1 Overview

In this section, we provided how we use deep Q-learning network algorithm to build the AutoPentest-DRL framework. The architecture has six different parts like Fig. 3.1 shows.

- *Network Scanning:* Integrate the scanning tools such as Nmap in order to find the target system's vulnerabilities
- *Topology Generation:* Produce multiple network topology to increase the network diversity
- *Training Dataset:* Use Shodan and other database to build our own training dataset
- *DQN Decision Engine:* Utilize the DQN to train our own model for choosing the attack path
- *Simplified Matrix:* Employ DFS algorithm to simplify the training matrix to ensure success rate
- *Metasploit RPC API:* Use RPC server to communicate with Metasploit to do the attack

From the left side we can see the blue line, it uses the logical network to verify the feasibility of the framework. The user first inputs the basic information which includes network topology, open port and existed vulnerability for every machine, the generated file will be input to the MulVAL to create the attack tree. Second the attack tree will be input to the DQN Decision Engine to calculate and choose the optimal attack path. Finally

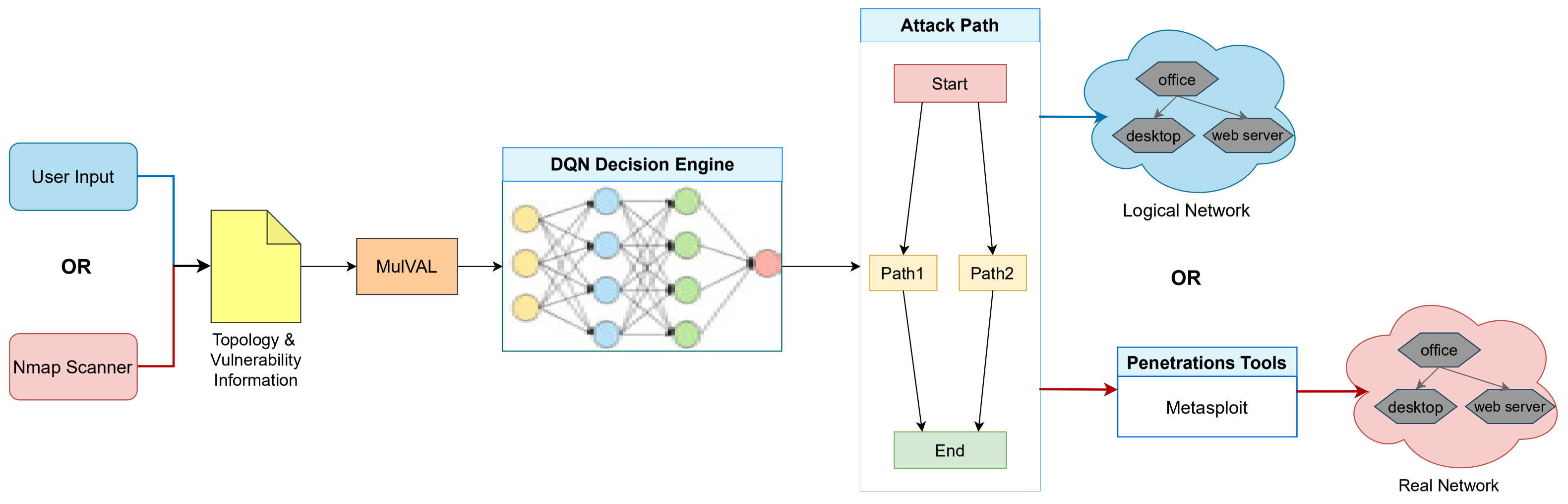


Figure 3.1: Architecture of the automated penetration testing framework.

the attack path will be imported to the logical network to see if it's correct. From the right side which means following the red line, the framework will be used in the real network. The middle part about attack path generation is basically the same, but the input is been replace by the Nmap scanning result. And the attack path will be imported to the penetration tools to do the penetration testing to the real network.

3.2 Network Scanning

To understand the inner traffic between the network topology, it may be related to the scanning tools. The aim of scanning tools is to discover the possible vulnerabilities, the excised ports and other useful information. Most of the scanning behaviors are not harmful, but some of them are executed with a malicious effect [29]. There are a great number of ports which include 65535 TCP ports and 65535 UDP ports.

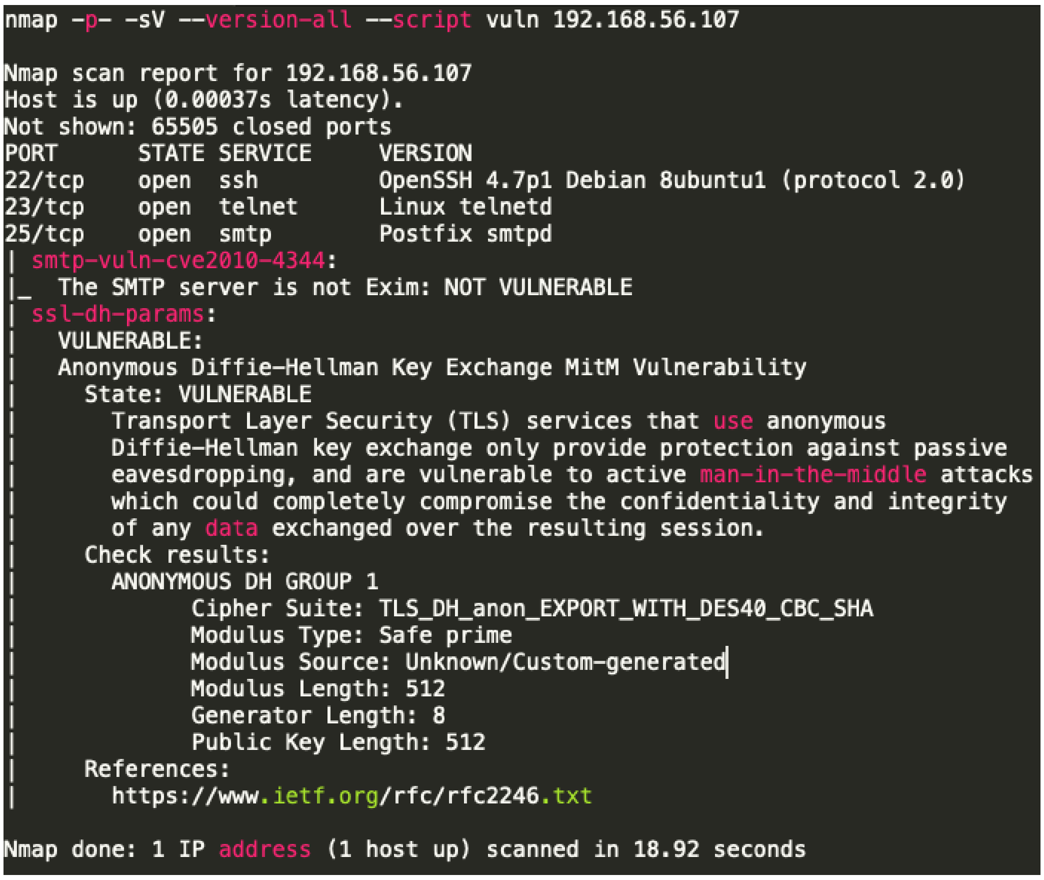
The representative of the scanning tools is Nmap, which is one of the most powerful port scanners in the world. With the assistance of the scanning of Nmap, we discover the detail information especially existing vulnerabilities of the target system according to the available ports and services. There are many options and scripts provided by Nmap in order to find out the target operation system, offerings and versions. For our framework, the most important thing is to figure out the existing vulnerability of the target node. NSE is one of the vulnerability scanning scripts which includes many vulnerability library files and network protocol library components. We can use the scan results of NSE to create an excellent vulnerability report simultaneously. Here we scan the target node by employing the following command:

```
# nmap -p --sV -oX --version-all --script vuln <ip>
```

The parameter of $-oX$ enables Nmap to output the scanning report in XML format. We extract the detailed information (including existing vulnerabilities, product name and so on) from the XML file for every target node and allocate them to the network topology file. The generated network topology will be used in the following steps. Fig. 3.2 is a Nmap scanning result report which includes all the needed information.

3.3 Topology Generation

Generally in machine learning, if we have more learning samples, we will get higher learning accuracy of the model. To improve the learning rate, we try to generate enough and sufficiently complex network topology models. There



```
nmap -p- -sV --version-all --script vuln 192.168.56.107

Nmap scan report for 192.168.56.107
Host is up (0.00037s latency).
Not shown: 65505 closed ports
PORT      STATE SERVICE     VERSION
22/tcp    open  ssh         OpenSSH 4.7p1 Debian 8ubuntu1 (protocol 2.0)
23/tcp    open  telnet      Linux telnetd
25/tcp    open  smtp        Postfix smtpd
| smtp-vuln-cve2010-4344:
|_  The SMTP server is not Exim: NOT VULNERABLE
| ssl-dh-params:
|   VULNERABLE:
|   Anonymous Diffie-Hellman Key Exchange MitM Vulnerability
|     State: VULNERABLE
|       Transport Layer Security (TLS) services that use anonymous
|       Diffie-Hellman key exchange only provide protection against passive
|       eavesdropping, and are vulnerable to active man-in-the-middle attacks
|       which could completely compromise the confidentiality and integrity
|       of any data exchanged over the resulting session.
|     Check results:
|       ANONYMOUS DH GROUP 1
|             Cipher Suite: TLS_DH_anon_EXPORT_WITH_DES40_CBC_SHA
|             Modulus Type: Safe prime
|             Modulus Source: Unknown/Custom-generated
|             Modulus Length: 512
|             Generator Length: 8
|             Public Key Length: 512
|     References:
|       https://www.ietf.org/rfc/rfc2246.txt

Nmap done: 1 IP address (1 host up) scanned in 18.92 seconds
```

Figure 3.2: Example of Nmap scanning result.

is a python tool named ”topology-generator” which is issued by *cesarghali* in 2015, it can generate random network topologies consisting of routers, clients, and servers. Input parameters can be specified in a configuration file. The generated topologies are written into JSON output files and optionally plotted in PDF format. Fig. 3.3 is a sample configuration file:

The parameter of ‘topologies’ means the number of generated topologies, ‘routers’ means the number of routers which can connect the servers and the clients, ‘router-links’ means the number of links between two routers, ‘clients’, ‘client-links’, ‘servers’ and ‘server-links’ are same as ‘routers’ and ‘router-links’. ‘channels’ is the number of channels used to connect the topology nodes. Each channel can have a different data rate. ‘channel-rates’ is a list of channel rates. The size of this list should match the value of the ‘channel’ parameter.

In order to run this tool, we can use the following command:

```
{
        "topologies": "1",
        "plot": "True",
        "dimensions": "10x10",
        "routers": "2",
        "router-links": "1-2",
        "clients": "5",
        "client-links": "1-2",
        "servers": "3",
        "server-links": "1-2",
        "channels": "1",
        "channel-rates": "10"
}
```

Figure 3.3: Sample configuration file of topology-generator.

```
# ./topo-gen.py -c <configFile> -o <outputFile>
```

$< configFile >$ is the configuration file path, $< outputFile >$ is the output file name without the extension. The bash command$topo - gen$ will create two output files: $< outputFile >$ *.json* and $< outputFile >$ *.pdf*.

Fig. 3.4 shows an example of generated network topology. The red points represent the routers, the green points represent the servers and the blue points represent the clients. The relationship between them is connected by the black line.

By defining those options we can create many kinds of different network topologies, in order to increase the diversity of the generated network topology, the start point and end point are been selected randomly. For example, suppose we take one of the green points as the attack starting point and one of the blue points as the final target point, by randomly selecting the starting point and the end point, we can generate a large number of different topological structures on the basis of one single network topology, thus greatly improving the diversity of the generated samples.

If we change one parameter in the configuration file, we can completely change the structure of the network topology. For example, if we change ‘client-links’ from ‘1-2’ to ’1’, the network topology is as Fig. 3.5 shows. So in this way, we can get enough complex network topology samples.

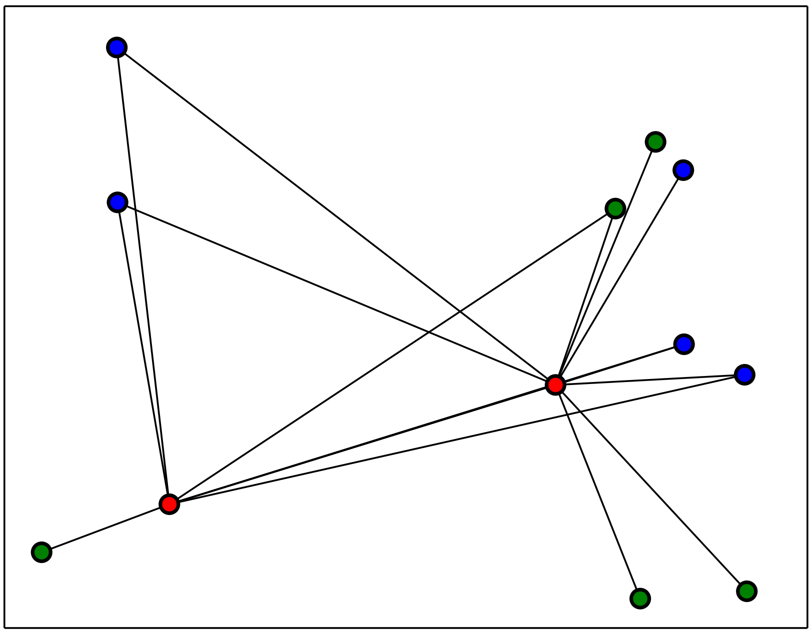


Figure 3.4: Example of generated network topology.

3.4 Training Dataset

The generation of training data is basically the same as described in [35]. It is been divided into 3 steps:

1. Shodan search engine is been employed to get the needed information of different hosts
2. MulVAL is been used to generate the attack trees for the selected network environment
3. Organize the above data into the format we need to suit for the DQN Decision Engine

Host Dataset

Host Dataset includes all the host information. In this process, Shodan search engine plays an important role. We need to use it to obtain real device information, so as to improve the correlation between the data and

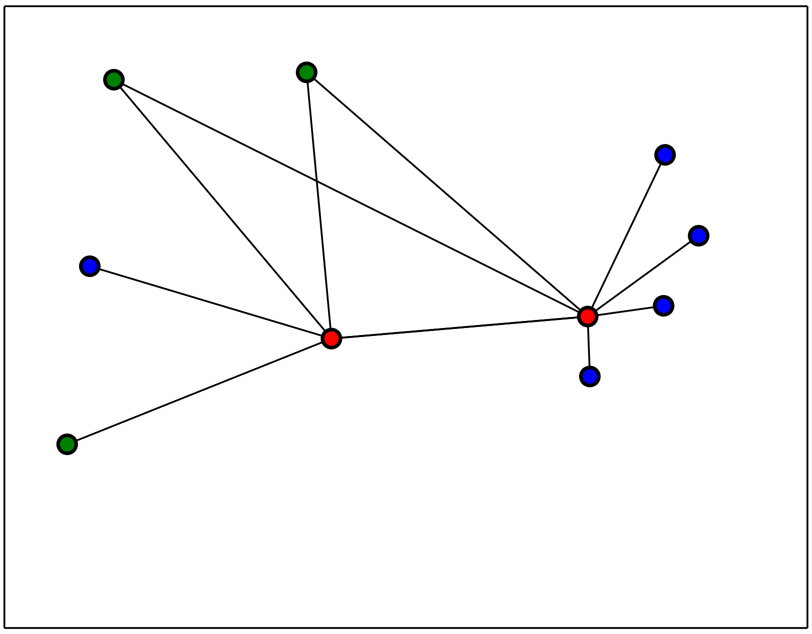


Figure 3.5: Example of alternative generated network topology.

the samples. Fig. 3.6 is an example of Shodan search engine. Here is a real mail server which is existed in the Internet. Some key information is been returned by the result of Shodan searching.

We generate a separate dataset file for every different service with the needed information. In order to protect the privacy of the device, the data collected by us only retains the non-sensitive information, and all the sensitive information is removed. Table 3.1 provides aprofile example of a mail server.

Table 3.1: Mail Server profile examples

Port	**Product**	**Protocol**	**OS**	**Vulnerability**
25	Haraka	smtp	CentOS	CVE-2012-1452
587	Exim	smtp	Ubuntu	CVE-2011-0419
995	Mailtraq	pop/imap	FreeBSD	CVE-2017-9617

```
   "info": "(Ubuntu)",
   "os": ubuntu 12,
   "port": 465,
   "product": "Apache James",
   "transport": "smtp",
   "version": "2.3.2",
   "vulns": {
      "CVE-2008-2049",
      ...
   }
```

Figure 3.6: Mail server information through Shodan search engine.

Vulnerability Dataset

In order to get the needed information of vulnerability, the vulnerability dataset is very necessary in our training process. We need to get the corresponding score through different vulnerabilities to get the reward. At present, the market mainly includes CVE and Microsoft vulnerability (MS) [36, 37]. In order to measure the different harm caused by different vulnerabilities, we usually use CVSS to evaluate them [38]. CVSS includes basic score and availability score. In this dataset, all the needed vulnerability information come from the National Vulnerability Database (NVD) and Microsoft. Table 3.2 is an example of the vulnerability dataset.

Table 3.2: Example of vulnerability dataset

MS ID	**CVE ID**	**Type**	**ExpScore**	**BaseScore**
MS12-039	CVE-2011-3402	Insufficient Information	8.6	9.3
MS11-069	CVE-2011-1970	Buffer Errors	10.0	5.0
MS16-124	CVE-2016-0070	Information Disclosure	8.6	4.3

3.5 Simplified Matrix

Simplifying the input matrix is the key step of generating training data. In the traditional deep reinforcement for attack tree, the neural network will learn how to find the most possible path in a huge matrix. It takes a considerable amount of time to adjust the learning parameters in order to obtain good training results. In our previous research [35], the accuracy rate is 0.863 and there is a possibility of improvement. At the same time, due to the expansion of our training data, the input matrix also becomes bigger and more complicated. In order these problems and improve the success rate, we propose to employ DFS to simplify the input matrix.

In this example attack tree like Fig. 2.6 shows, the start point of an attacker is the $Internet$ which is located in node 33, and the final objective of the attacker is to have an access to the $Workstation2$ which is located in node 1. We suppose an attacker can move in any direction until he reaches the final goal. In this case we need to give every node a reward score to make a transfer matrix:

- For every node which exists vulnerability, we use the $Score_{vul}$ to represent the reward.
- For the $execCode$ node except node 1, the reward score is 0.2.
- The reward value of start node 33 is 0.01 and the end node 1 is 100.
- For any other nodes, we give the reward score 0, and if the node cannot access another node, the reward score is -1. Because we have 57 nodes, so the size of the matrix is 57 × 57.

The transfer matrix is as follows:

$$\begin{bmatrix} -1. & -1. & -1. & -1. & -1. & \cdots & -1. \\ 100. & -1. & -1. & -1. & -1. & \cdots & -1. \\ -1. & 0.2. & -1. & -1. & -1. & \cdots & -1. \\ -1. & -1. & 0.01. & -1. & -1. & \cdots & -1. \\ -1. & 4.5. & -1. & -1. & -1. & \cdots & -1. \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ -1. & -1. & -1. & -1. & -1. & \cdots & -1. \\ -1. & -1. & -1. & 0.2. & -1. & \cdots & -1. \end{bmatrix}$$

The transfer matrix indicates all of the actions and we need to simplify it in order to send it to the DQN model and save time. For this purpose, DFS is considered to be integrated into the framework to simplify the transfer matrix. Firstly we search all of the different attack paths. Next we try to generate a new matrix according to this format $[start, midsteps, goal]$. $start$ means the reward score of the start position, $goal$ means the reward score of the goal position, *midsteps* is the total of the intermediate steps. The size of the simplified matrix is 10×10 and it is as follows:

$$\begin{bmatrix} 0.01. & 4.6. & 2.5. & 1.6. & \cdots & -1. \\ 0. & -1. & -1. & -1. & \cdots & 100. \\ 0. & -1. & -1. & -1. & \cdots & 100. \\ 0. & -1. & -1. & -1. & \cdots & 100. \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0. & -1. & -1. & -1. & \cdots & 100. \\ -1. & 0. & 0. & 0. & \cdots & -1. \end{bmatrix}$$

The state of the attacker is displayed by the matrix pattern. For example, if the current attacker is in row 1(state 1), the value $[0.01, 4.6, 2.5, 1.6, ..., -1]$ means if we move to row 2(state 2), we will get the score of 4.6. Since the current attacker moves to state 2, the value of the current state is $[0, -1, -1, -1, ..., 100]$.

3.6 DQN Decision Engine

The DQN Decision Engine is the core of the framework, it will be learned many times and output the correct attack path [35]. Fig. 3.7 shows the relationship between $DQNDecisionEngine$ and input data $TrainingData$ and output data $AttackPath$.

In the DQN Decision Engine, the input is the simplified matrix, the output is the most possible attack tree, and the $softmax$ is been selected as the active function. The DQN Decision Engine also consists of an agent and states, the agent can move to next state according to the action. Taking an action in a state result in a value reward.

During the process of learning, the agent of DQN Decision Engine represents a real attacker, the simplified matrix is the environment. The agent can move freely in the matrix regardless of the values in the matrix. The ultimate goal is to move to the nodes we need which means we find the target successfully. The value in the matrix represents the reward of each move step, it is defined by ourselves. In the set of rewards, each vulnerability is very important. According to the Common Vulnerability Scoring System (CVSS), we get their rewards by calculating as the following formula 3.1. We use Matrix R to create Matrix Q for the goal and employ Q-learning to update the learning policy by the following equation 3.2 [21].

$$Score_{vul} = baseScore \times \frac{exploitablityScore}{10} \tag{3.1}$$

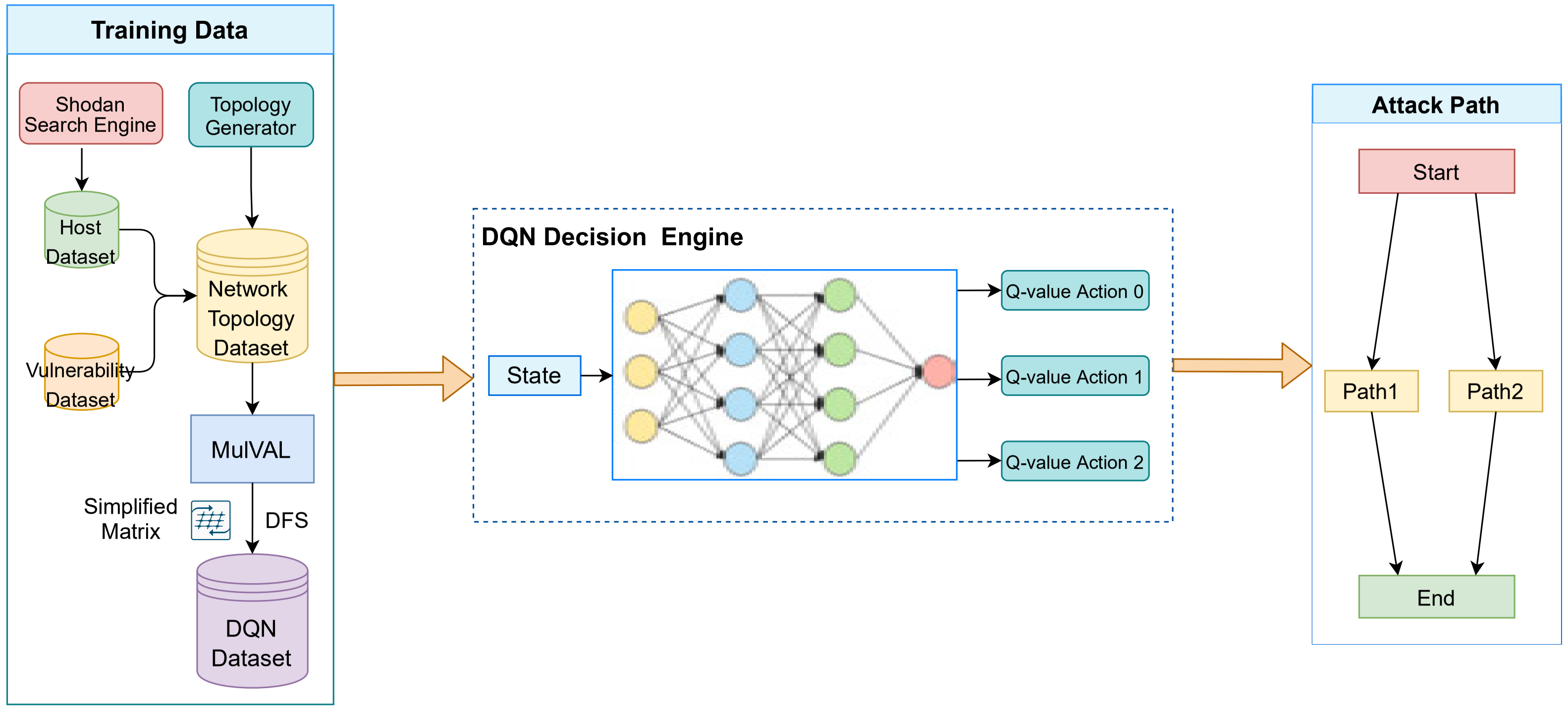


Figure 3.7: Overview of DQN Decision Engine.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{3.2}$$

In this equation, the target of Q-learning is $R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)$, S is a set of states and A is a set of actions, t is the time. It uses $\epsilon - greedy$ to generate action a_{t+1}, but the action used in order to calculate the target is the largest action for $Q(S_{t+1}, a)$. Through continuous learning of the parameters, the model can accumulate a lot of experience. Details of Deep Q-learning algorithm about DQN Decision Engine are described in the following algorithm 1:

Algorithm 1: DQN Decision Engine Algorithm

```
Input: Simplified State/Reward (Environment) Matrix (R),
       Discounted factor ($\gamma$), Learning rate ($\alpha$)
Output: The most possible action selection policy (Matrix Q)
1  for episode=1, M do
2      Initlalize $Q$ with random weights for t=1, T do
3          Select a random state ($S_t \in S$)
4          while current state != goal state do
5              Select a random action $a_t$ from all possible actions,
6              otherwise select $a_t = max_a Q^*(S_t, a; w)$
7              Execute the action $a_t$ and observe the reward $r_t$
8              Update Q with the transition:
9              $Q(S_t, A_t) \leftarrow$
                 $Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$
10             Current state = next state
```

The *Input* is the simplified matrix which includes state, reward and so on. The *Output* is the most possible action selection policy that is called 'Matrix Q'. First select a state randomly, then judge whether *currentstate* and *currentstate* are equal. If they are not equal, select an action randomly, execute it and get the reward. Next use *transition* to update the *Q*. After looping many times, *currentstate* is equal to the *currentstate*, which means that the algorithm finds the most possible path successfully.

3.7 Metasploit RPC API

The RPC API of Metasploit enables to communicate with the Metasploit employing HTTP-based remote procedure call (RPC) services. In order to use it, you must start the RPC server by loading the msgrpc plugin. There is an interface named MessagePack which comes from the msgrpc plugins, it will open a port to listen the input commands. In this way, you can send information to Metasploit and also get the return results through the port. After starting the RPC server, you can use msfrpc utility to connect it by running the following code:

```
# ruby msfrpc -U <username> -P <password> -a <ip address>
```

$< username >$ and $< password >$ are the server's username and password to access msfrpcd. After connecting the RPC server, you will get the following response:

```
[*] exec: ruby msfrpc -U xxx -P xxx -a x.x.x.x
[*] The 'rpc' object holds the RPC client interface
[*] Use rpc.call('group.command') to make RPC calls
```

Fig. 3.8 shows the entire attack process of using Metasploit RPC API. Fisrtly DQN Decision Engine will send some information that includes attack path for the attack tree, the attack tree will be converted to a format which can be recognized by RPC server through the middle ware module; Secondly the Metasploit RPC api server will send attack commands to the penetration tools like Metasploit to tell it how to attack the target server; Finally the penetration tools will so the penetration testing to the test server to achieve the target.

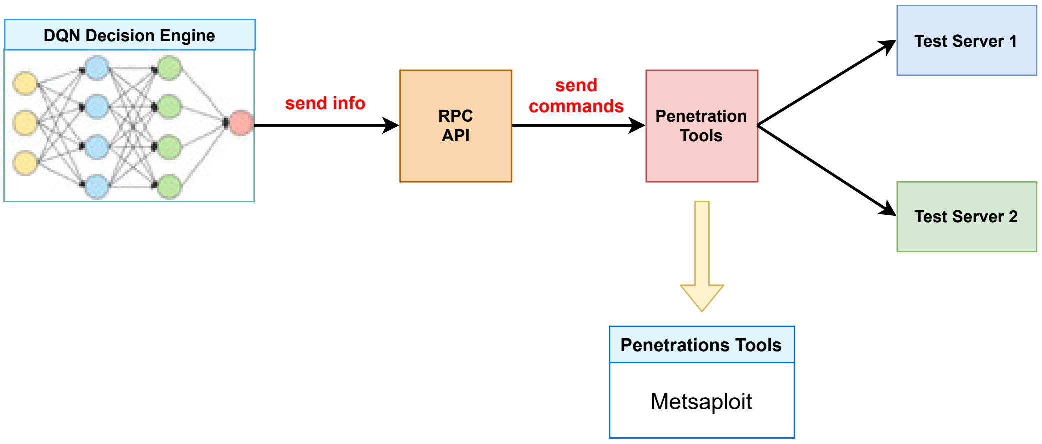


Figure 3.8: Attack process of using Metasploit RPC API.

In order to convert the output path by DQN Decision Engine into the format which can be recognized by Metasploit, we need to extract the useful information in the path. For example, in the path "33 → 31 → 24 →

$23 \rightarrow 22 \rightarrow 20 \rightarrow 17 \rightarrow 16 \rightarrow 15 \rightarrow 13 \rightarrow 10 \rightarrow 9 \rightarrow 8 \rightarrow 6 \rightarrow 5 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1$" (cf. Fig. 4.2), the node 8 means that the fileServer will suffer from *execCode* attack. The node 29 describes the detailed information about the existing vulnerability (CVE-2015-3306) in Apache product. If we want to get the root privilege in fileServer, we need to employ the vulnerability of CVE-2015-3306 to penetrate the fileServer. Other nodes about servers or workstations are in the same situation. Based on the above characteristics, we simplified the entire attack process into the following path: "$33 \rightarrow 22 \rightarrow 15 \rightarrow 8 \rightarrow 1$". In this way, penetration tools like Metasploit only need to know the IP address and vulnerabilities of every machine to achieve automated penetration testing.

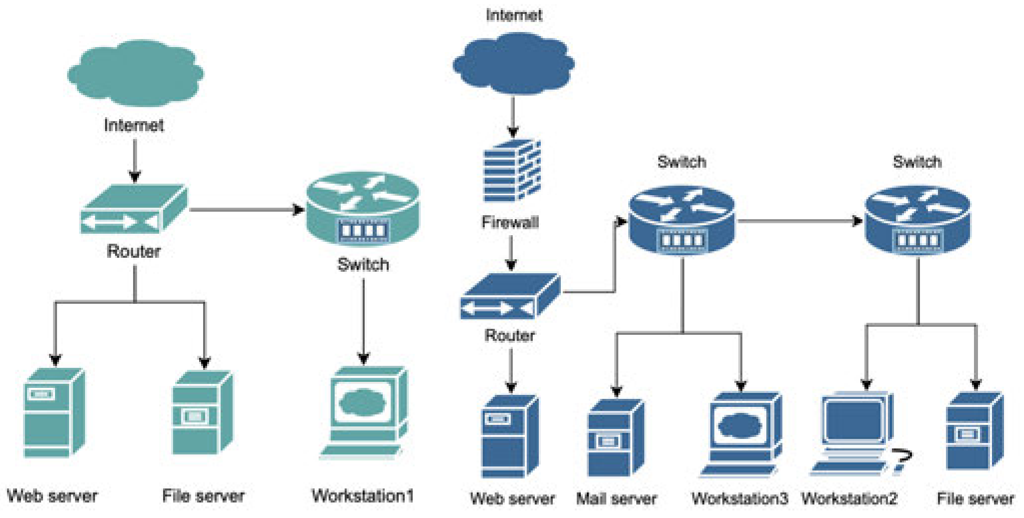
Chapter 4

Experiment Results and Discussion

In this chapter, we build an experiment to train the DQN Decision Engine. At the same time, we use some common network topologies to evaluate our framework in a logical network. At the same time, we also build a virtual machine environment to execute real penetration testing in order to test the validity of the framework on a real network.

4.1 Network Topology Models

Fig. 4.1 provides two different network topologies. The ‘Topology 1’ of Fig. 4.1(a) is a simple network which is a personal home network environment. It includes the ‘Web server’, ‘File server’ and ‘Workstation 1’. The ‘Topology 2’ of Fig. 4.1(b) on the right side is a company’s network environment. It is a bit more complicated than the home network topology and also includes some computer machines. ‘Web server’ is been used to show the website of the company through HTTP/HTTPS protocol. ‘Mail server’ is been set to receive and deliver emails. ‘File server’ offers a central storage place for files on internal data media. Here we mainly talk about the ‘Topology 2’ because it’s more realistic than ‘Topology 1’.



(a) Topology 1 (b) Topology 2

Figure 4.1: Experiment network topology models.

In this experiment network topology, the different machines are located in the different areas. For example in ‘Topology 2’, ‘Web server’ directly connects with the router which is located behind the ‘Firewall’. ‘Mail server’ and ‘Workstation 3’ are located in a subnet that is linked by a router with the ‘Firewall’. ‘Workstation 2’ and ‘File server’ are in another subnet that connects the subnet of the ‘Web server’.

For every node in the attack path, we have formulated the following connection rules:

- The ‘Internet’ is the start point that the attacker has access to the ‘Web server’ through HTTP and HTTPS.
- Through file transfer protocols like FTP, NFS, the ‘File server’ and the ‘Web server’ have a connection to each other.
- Through file transfer protocols like FTP, NFS, the ‘Workstation 2’ and the ‘File server’ have a connection to each other.
- Through HTTP and HTTPS protocols, the ‘Workstation 2’ and ‘Workstation3’ can connect to the ‘Internet’.
- Through IMAP and SMTP protocols, the ‘Workstation 3’ has a connection to the ‘Mail server’.

4.2 DQN Dataset Generation

For every device, we need to initialize detailed information such as vulnerabilities, open ports, products, and protocols. All those information comes from the training data which is mentioned in Section 3.4. Table 4.1 provides some information about these machines for ‘Topology 2’.

Table 4.1: Network information for Topology 2

Host	Vulnerability	Product	Port	Protocol
Web server	CVE-2017-2619	Samba	139	SMB
Mail server	CVE-2017-0307	-	465	SMTP
File server	CVE-2007-6232	-	21	FTP
Workstation3	-	-	-	HTTP/HTTPS/SMTP
Workstation2	-	-	-	HTTP/FTP

As for the vulnerability dataset, there is also some needed information. For instance, in the Table 4.2, the vulnerability of ‘CVE-2019-0211’ needs its own type and caused effect which is ‘Privilege Escalation’ and ‘root’. And for the ‘CVE-2015-3306’, the effect is been changed to ‘user’ which means you can not execute some high privilege actions.

Table 4.2: Information of different vulnerabilities

Vulnerability	Effect	Type
CVE-2015-3306	user	Improper Access Control
CVE-2019-0211	root	Privilege Escalation
CVE-2020-17087	root	Privilege Escalation

Given the network information of ‘Topology 2’, we use MulVAL to generate an attack tree which is shown in Fig. 4.2. Every node has its own description, for example the node ‘35’, it means that there is a vulnerability named ‘CVE-2011-4718’ existed in the ‘fireWall’, which can cause attackers to obtain root privilege. In the scenario, the start point of the attacker is the node ‘33’ Internet, the target is to execute malicious code on the node ‘1’ which means we have access to the final target workstation.

After we get the generated attack tree, we need to create the DQN dataset by combining the attack tree with the host dataset and vulnerability dataset.

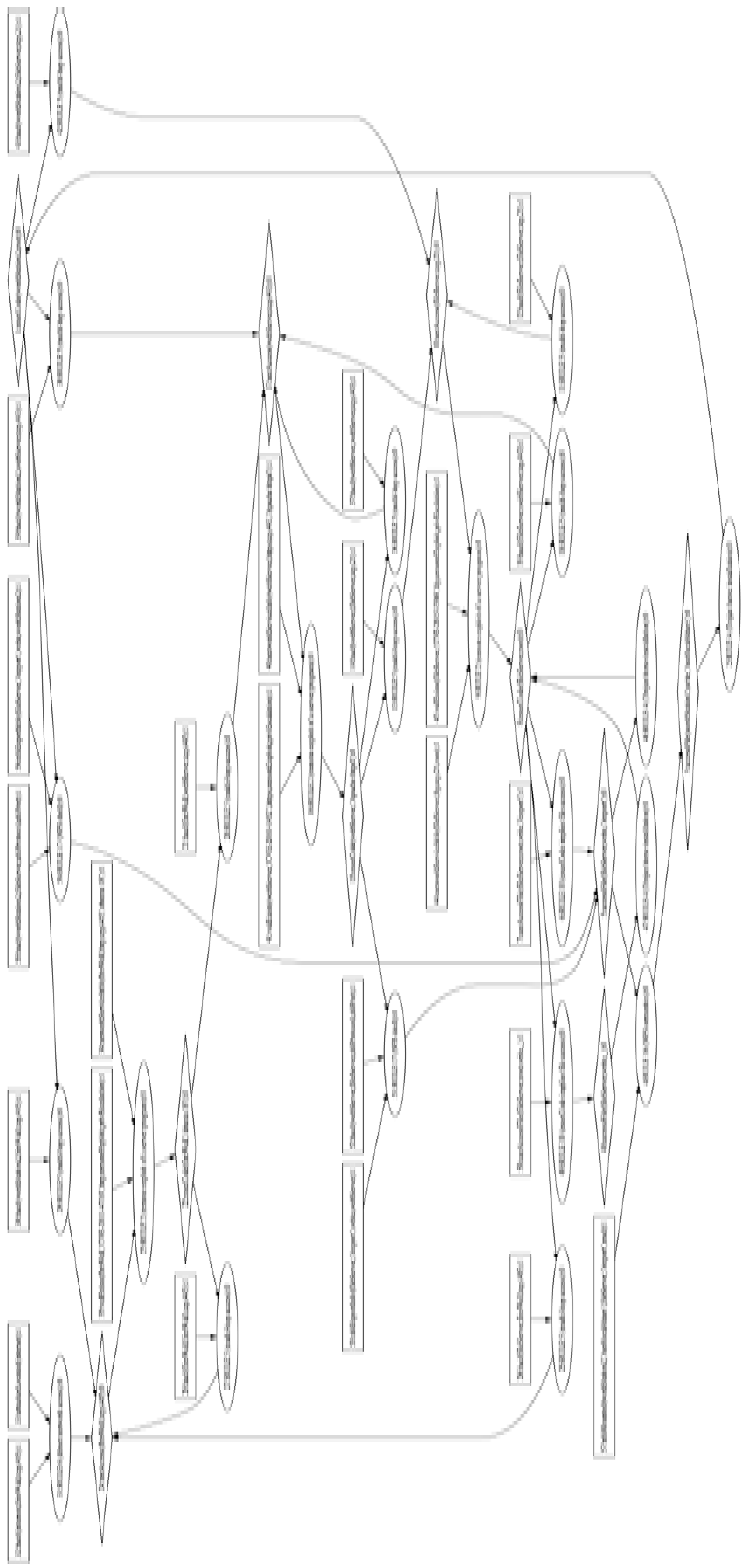
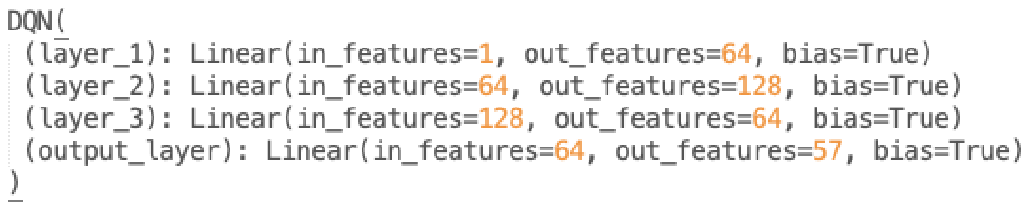


Figure 4.2: Generated attack tree.

We transfer the attack tree to the matrix in order to use it as the input of the neural network, then the DFS algorithm is been used to simplify the matrix where is been talked in Section 3.5. In this way we can generate the input data of DQN Decision Engine which is called DQN dataset.

4.3 DQN Training Results

After we generate the needed dataset, we need to train the DQN model. In this experiment scenario, we select a neural network with 3 layers. The network model we used in the experiment scenario is provided in Fig. 4.3. In the neural network, the input is the DQN dataset which is transferred to matrix, the output is the most possible attack path. In order to verify, we define that the most possible path in the matrix is the path with the minimum steps while choosing the biggest value of reward.



```
DQN(
 (layer_1): Linear(in_features=1, out_features=64, bias=True)
 (layer_2): Linear(in_features=64, out_features=128, bias=True)
 (layer_3): Linear(in_features=128, out_features=64, bias=True)
 (output_layer): Linear(in_features=64, out_features=57, bias=True)
)
```

Figure 4.3: DQN network model in the experiment scenario.

Table. 4.3 shows an example of the different attack paths for the target end node 1. As we can see, there are two example attack paths which the first one uses all of the existed vulnerabilities and the second one uses two different vulnerabilities. If we start from the node 33 and the end node is node 1, the maximum rewards of the first one are bigger than the second one, which means that our training has worked, the strategy will of DQN Decision Engine prioritize to select the attack path with the largest reward.

Table 4.3: Different attack paths for the end node 1

Start Node	**End Node**	**Attack Path**	**Maximum Rewards**
33	1	33→22→15→8→1	113.5
33	1	33→22→15→5→1	108.7

Fig. 4.4 shows the running information of DQN model through the representation of enterprise network state space. We change the discount factor

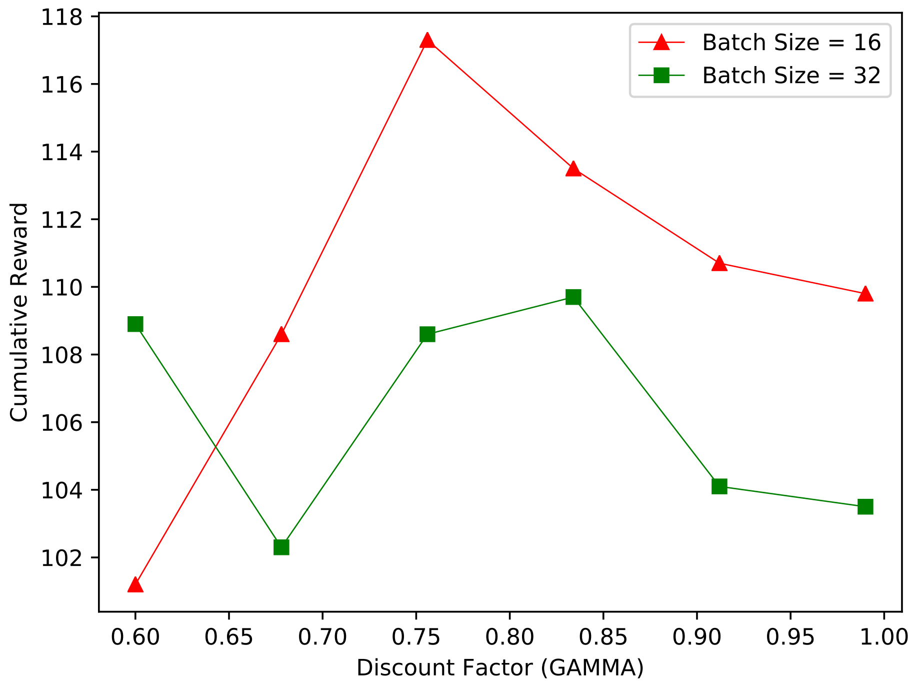


Figure 4.4: Cumulative reward changes with discount factor (GAMMA).

from $GAMMA = 0.6$ to $GAMMA = 0.99$. The lower value of the discount factor indicates that the reward of the terminal state has no substantial impact on the reward of the current state, while the higher value of the discount factor indicates that the higher reward of the terminal state will affect the reward of the agent in the current state. We also use different batch sizes (BS) to observe the effect on learning the best strategy. When $BS = 16$, the proxy achieves faster convergence compared with $BS = 32$. So $BS = 16$ is been chosen as our batch size.

The variation of the average reward with the number of the epoch is shown in Fig. 4.5. There are about 150 epochs, the reward is small at first, as the number of epoch increases, the reward is getting bigger. After about 80 iterations, the increasing trend of reward begins to gradually slow down until stable. At this time, the DQN model begins to stabilize which means it finds the best strategy in the current state. In this case of 'Topology 2', the attack path is "$23 \rightarrow 21 \rightarrow 20 \rightarrow 19 \rightarrow 18 \rightarrow 16 \rightarrow 15 \rightarrow 14 \rightarrow 13 \rightarrow 11 \rightarrow 10 \rightarrow 9 \rightarrow 8 \rightarrow 6 \rightarrow 5 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1$" (cf. Fig. 4.7), which means that the DQN network model chooses the strategy that exploits the vulnerabilities as much as possible.

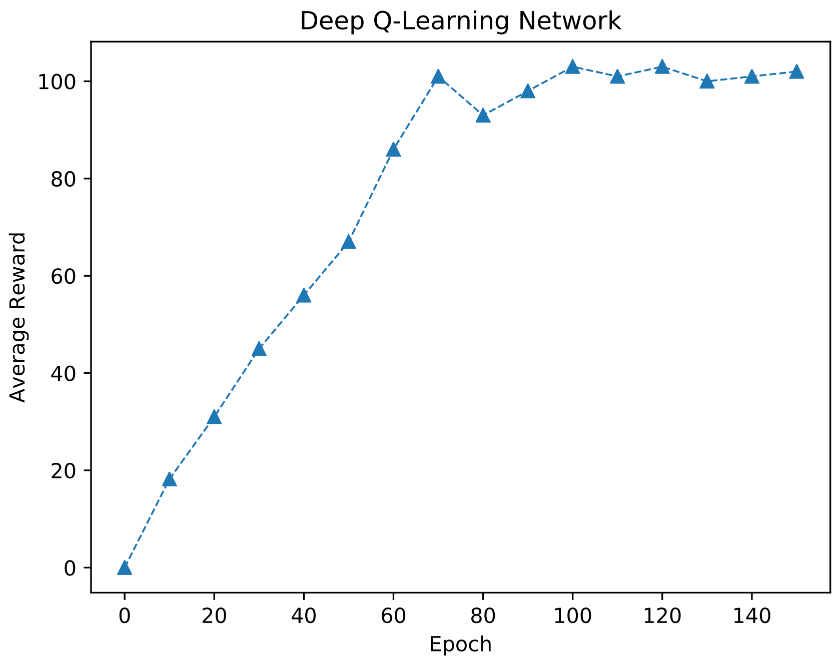


Figure 4.5: Average reward change with the increase of epoch.

Fig. 4.6 is the error ratio of the DQN network model, we can find that this framework has a good performance in convergence. The error rate gradually decreases and stabilizes as the number of iterations increases, that the overall trend is opposite to Fig. 4.5. Therefore, our DQN model has better adaptability.

4.4 Evaluation on Logical Network

In order to evaluate the framework on the logical network, we adopt 1000 different attack trees as our validation data to test the performance. 500 samples are generated by the 'topology-generator' according to 'Topology 1', another 500 samples are generated according to the 'Topology 2'. For the purpose of distinguishing between correct and wrong attack paths, we use the special generated attack path as our evaluation standard. There is a parameter named '-p' in the MulVAl that uses deep trimming algorithm to generate the attack trees like Fig. 4.7 shows. The node 23 which is marked by blue circle is the start point, the node 1 which is marked by a green circle

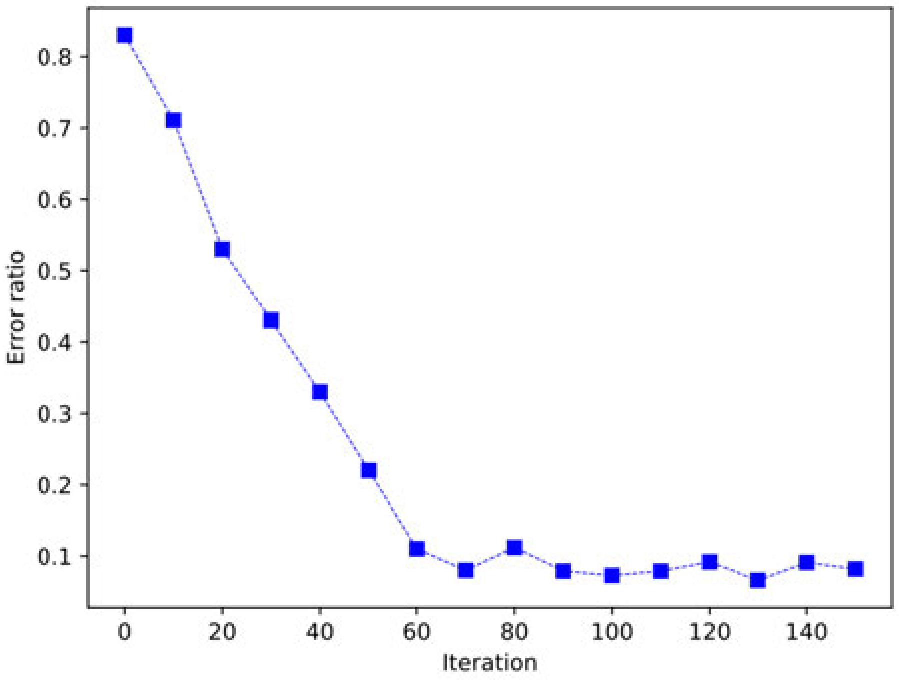


Figure 4.6: Error ratio of the DQN network model.

is the end point. The other that is marked by a red circle are the middle points.

If you look closely at the attack tree, there are two different attack paths. One chooses to exploit the vulnerability to infiltrate the 'File server', and the other chooses to use the NFS shell. In this experiment, since the NFS shell is difficult to use, in order to improve the success rate, we default to choose the attack path with more vulnerabilities as the correct attack path.

The result is shown in the Fig. 4.8. There are totally 932 samples that find the correct path. For the 'Topology 1', the number of correct paths and wrong path are 473 and 27, the accuracy is 0.946. For the 'Topology 2'. the number of correct paths and wrong path are 459 and 41, the accuracy is 0.918. So the average accuracy is 0.932. The result is shown that this framework has a good performance in distinguishing the attack path. Since the deep reinforcement learning framework contains a multi-layer neural network model, the neural network has a certain generalization ability for data from a data-driven perspective, and it is robust from a physical system perspective.

In order to show the advantage of our DQN Decision Engine, we build

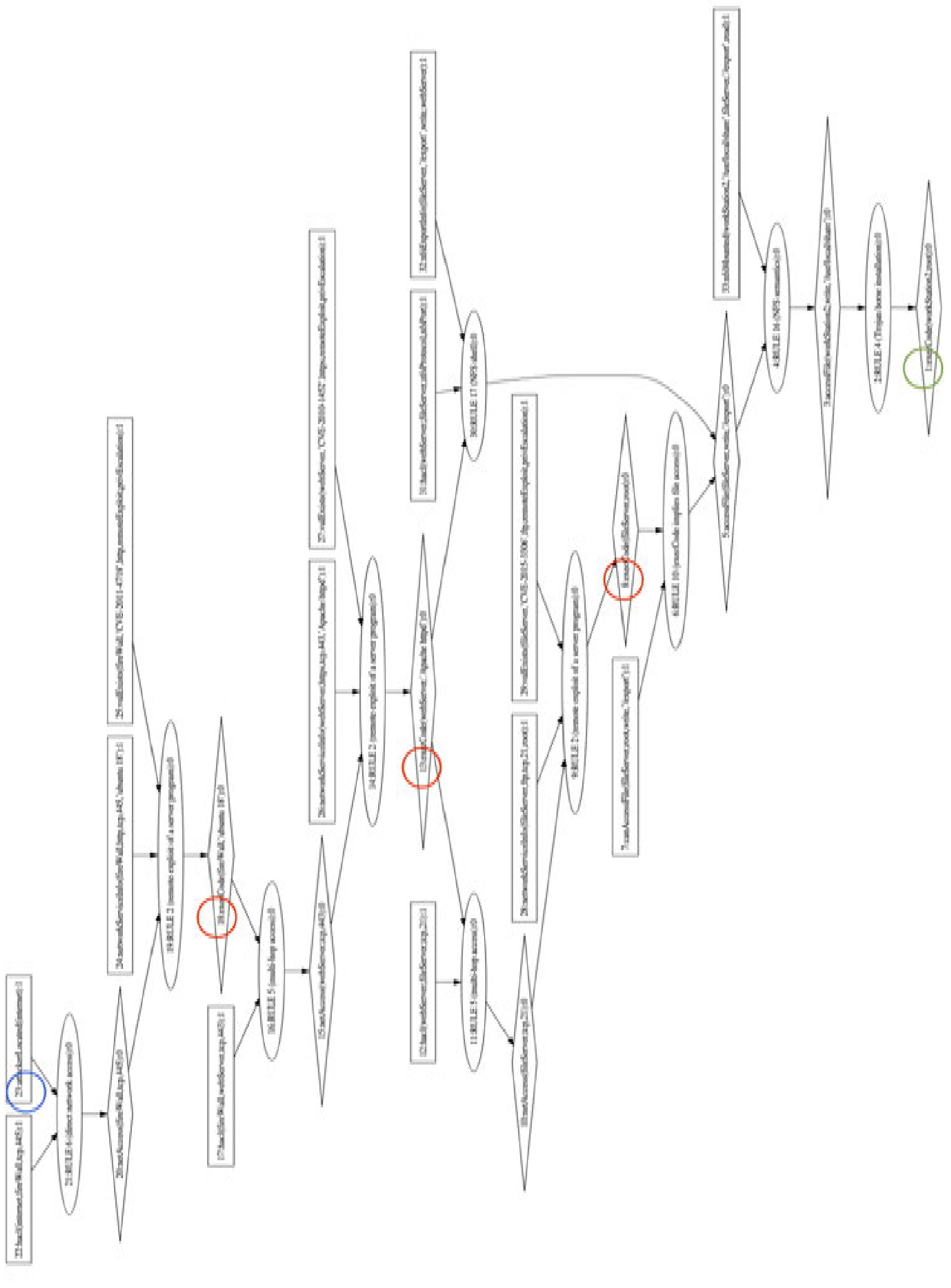


Figure 4.7: Attack tree generated by deep trimming.

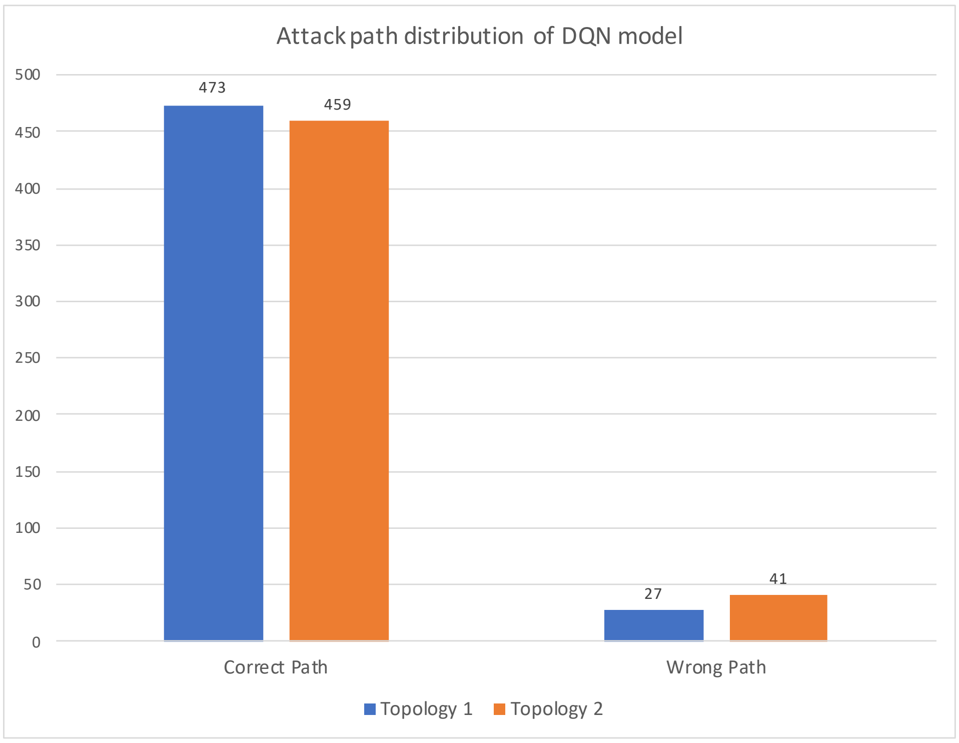


Figure 4.8: Attack path distribution of DQN model.

cooperation with the other related approaches. Table 4.4 provides comparative analysis from three main aspects. Our AutoPentest-DRL framework has a leading advantage in dealing with the network environment on the three characteristics.

Table 4.4: Comparison of different methods

Characteristics	PDDL	FF	AutoPentest-DRL
Re-planning	○	○	○
Uncertain path		○	○
Multi-host			○

4.5 Evaluation on Real Network

In order to evaluate the effectiveness of this framework in the real network environment, we build a real network topology environment using virtual machines. The basic network topology is generated as 'Topology 2' of Fig. 4.9 shows.

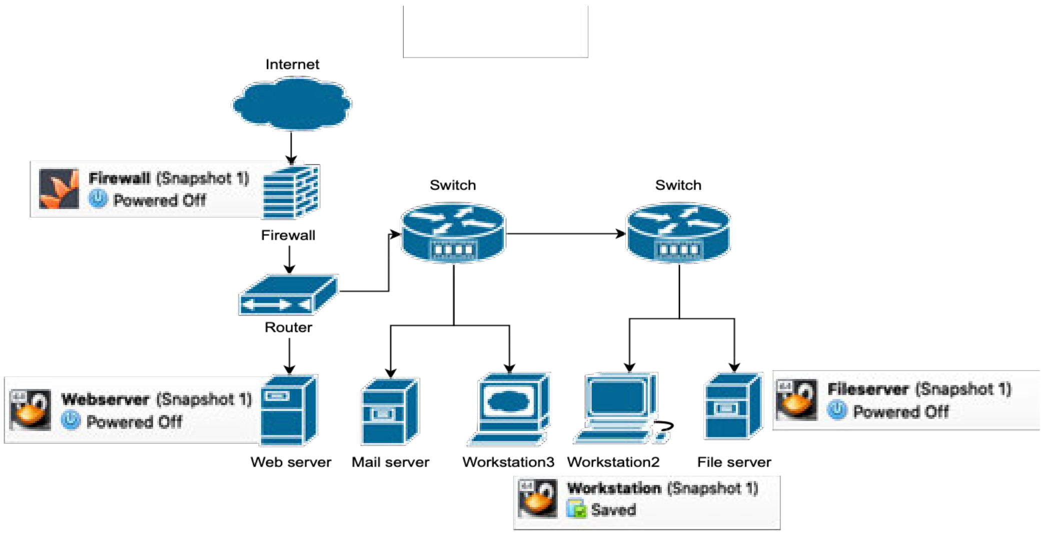


Figure 4.9: Real network environment.

The 'Firewall' is run in 'ubuntu 12', the 'Web server' is run in 'ubuntu 16', the 'Mail server' is run in 'ubuntu 16', the 'File server' is run in 'ubuntu 12', the 'Workstation1' and 'Workstation2' are both run in 'ubuntu 18'. We use three different vulnerabilities (CVE-2007-2447, CVE-2010-2075, CVE-2011-2523) and one test Trojan (generated by Metasploit).

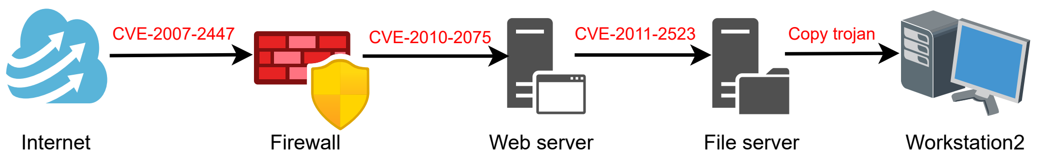


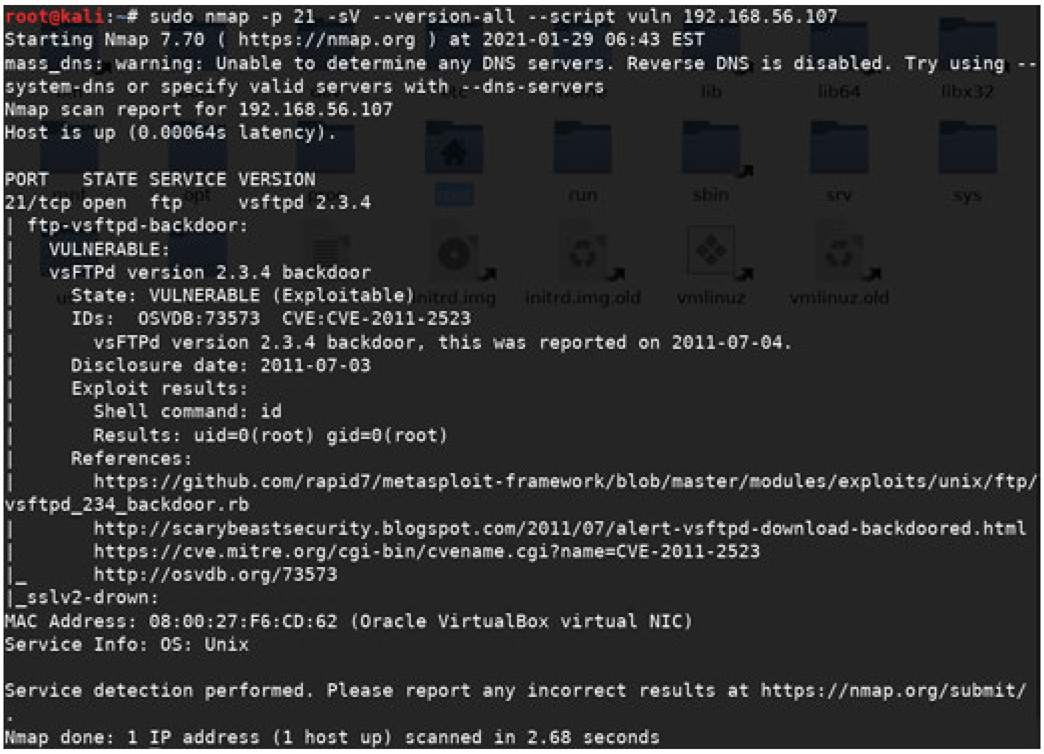
Figure 4.10: Penetration testing process for an experiment scenario.

The three virtual machines (Firewall, Web server, File server) are separately been inserted into the above vulnerabilities like Fig. 4.10 shows.

The detailed penetration testing steps are shown as follow:

1. Nmap is been used to obtain vulnerability information of each machine in the network environment.

2. MulVAL generates attack trees by combining Nmap scanning results with the configuration file which contains network topology information.



```
root@kali:~# sudo nmap -p 21 -sV --version-all --script vuln 192.168.56.107
Starting Nmap 7.70 ( https://nmap.org ) at 2021-01-29 06:43 EST
mass_dns: warning: Unable to determine any DNS servers. Reverse DNS is disabled. Try using --
system-dns or specify valid servers with --dns-servers
Nmap scan report for 192.168.56.107
Host is up (0.00064s latency).

PORT   STATE SERVICE VERSION
21/tcp open  ftp     vsftpd 2.3.4
| ftp-vsftpd-backdoor:
|   VULNERABLE:
|   vsFTPd version 2.3.4 backdoor
|     State: VULNERABLE (Exploitable)
|     IDs:  OSVDB:73573  CVE:CVE-2011-2523
|       vsFTPd version 2.3.4 backdoor, this was reported on 2011-07-04.
|     Disclosure date: 2011-07-03
|     Exploit results:
|       Shell command: id
|       Results: uid=0(root) gid=0(root)
|     References:
|       https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/unix/ftp/
vsftpd_234_backdoor.rb
|       http://scarybeastsecurity.blogspot.com/2011/07/alert-vsftpd-download-backdoored.html
|       https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-2523
|_      http://osvdb.org/73573
|_sslv2-drown:
MAC Address: 08:00:27:F6:CD:62 (Oracle VirtualBox virtual NIC)
Service Info: OS: Unix

Service detection performed. Please report any incorrect results at https://nmap.org/submit/
.
Nmap done: 1 IP address (1 host up) scanned in 2.68 seconds
```

Figure 4.11: Nmap scanning result of File server.

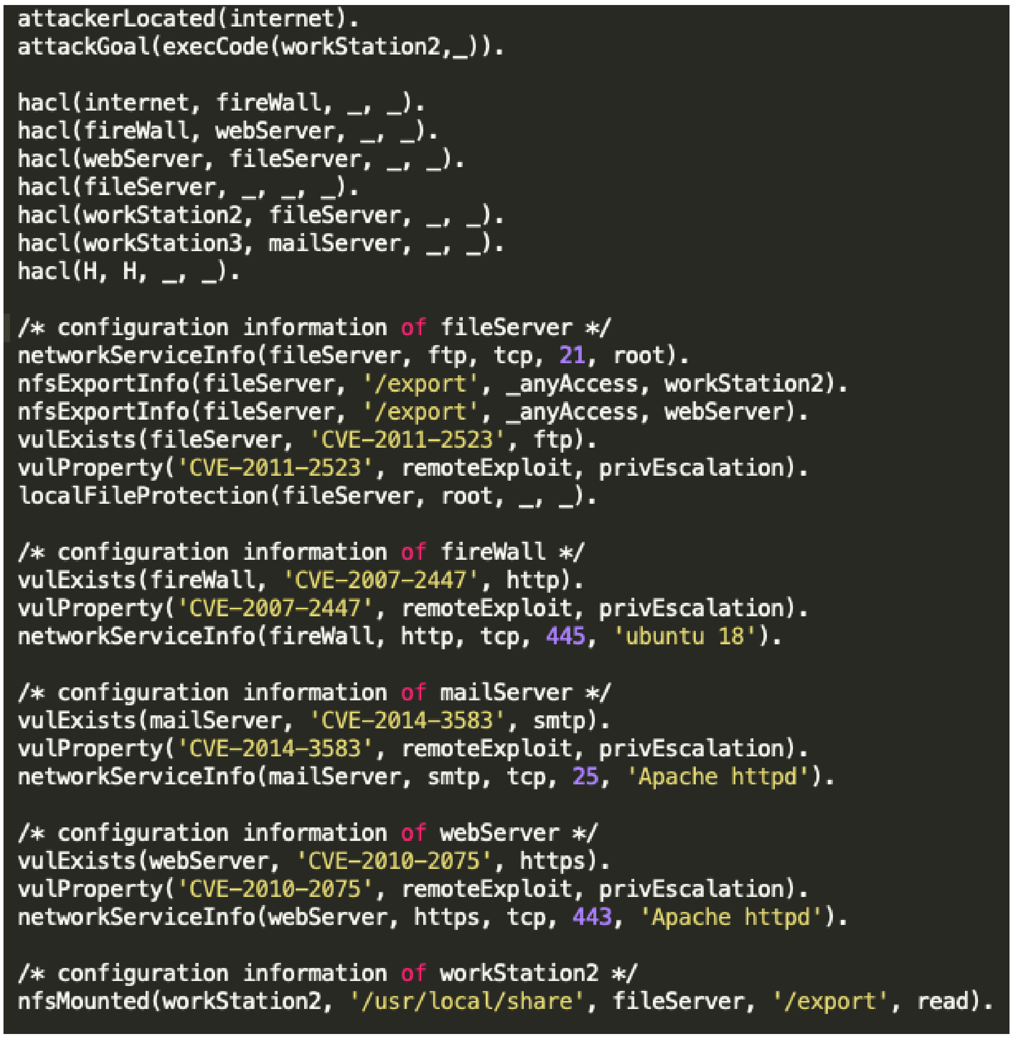
3. Attack tree is been transformed to the matrix and sent to the DQN Decision Engine.

4. DQN Decision Engine calculates and choose the optimal attack path and sent it to the Metasploit RPC server.

5. Metasploit RPC server sends corresponding attack command to the test server to do the penetration testing.

The Nmap scanning result of 'File server' of Step 1 is been provided in Fig. 4.11. Here we successfully find the vulnerability of CVE-2011-2523 which the type is 'ftp-vsftpd-backdoor' and the port is 21. The other two machines (Firewall and Web server) are as the same.

The configuration file of Step 2 is shown in Fig. 4.12. As we can see, the parameter of 'attackerLocated' is the start point, 'attackGoal' is the end point, 'hacl' is been used to define the network topology, 'networkServiceInfo' defines the basic information of this machine, 'vulExists' means what kind of vulnerabilities exist on this machine, 'vulProperty' is the property of the vulnerability. For this network topology and vulnerability information, the generated attack tree is as Fig. 4.13 shows.



```
attackerLocated(internet).
attackGoal(execCode(workStation2,_)).

hacl(internet, fireWall, _, _).
hacl(fireWall, webServer, _, _).
hacl(webServer, fileServer, _, _).
hacl(fileServer, _, _, _).
hacl(workStation2, fileServer, _, _).
hacl(workStation3, mailServer, _, _).
hacl(H, H, _, _).

/* configuration information of fileServer */
networkServiceInfo(fileServer, ftp, tcp, 21, root).
nfsExportInfo(fileServer, '/export', _anyAccess, workStation2).
nfsExportInfo(fileServer, '/export', _anyAccess, webServer).
vulExists(fileServer, 'CVE-2011-2523', ftp).
vulProperty('CVE-2011-2523', remoteExploit, privEscalation).
localFileProtection(fileServer, root, _, _).

/* configuration information of fireWall */
vulExists(fireWall, 'CVE-2007-2447', http).
vulProperty('CVE-2007-2447', remoteExploit, privEscalation).
networkServiceInfo(fireWall, http, tcp, 445, 'ubuntu 18').

/* configuration information of mailServer */
vulExists(mailServer, 'CVE-2014-3583', smtp).
vulProperty('CVE-2014-3583', remoteExploit, privEscalation).
networkServiceInfo(mailServer, smtp, tcp, 25, 'Apache httpd').

/* configuration information of webServer */
vulExists(webServer, 'CVE-2010-2075', https).
vulProperty('CVE-2010-2075', remoteExploit, privEscalation).
networkServiceInfo(webServer, https, tcp, 443, 'Apache httpd').

/* configuration information of workStation2 */
nfsMounted(workStation2, '/usr/local/share', fileServer, '/export', read).
```

Figure 4.12: Configuration file.

Follow the above steps to achieve a complete automated penetration testing process. Through comparison and analysis, we found that this framework has a higher success rate in the network environments with a small network topology structure like Fig. 4.1, and is able to complete automated penetration testing according to the principle of adopting exploit vulnerabilities as much as possible.

Fig. 4.10 shows the entire automated penetration testing process, the attacker uses three different vulnerabilities to attack three servers in order to get the control authority, and finally transmit the test Trojan from the ‘Internet’ to the ‘Workstation2’.

Through the vulnerability of CVE-2010-2075, we get the root privilege

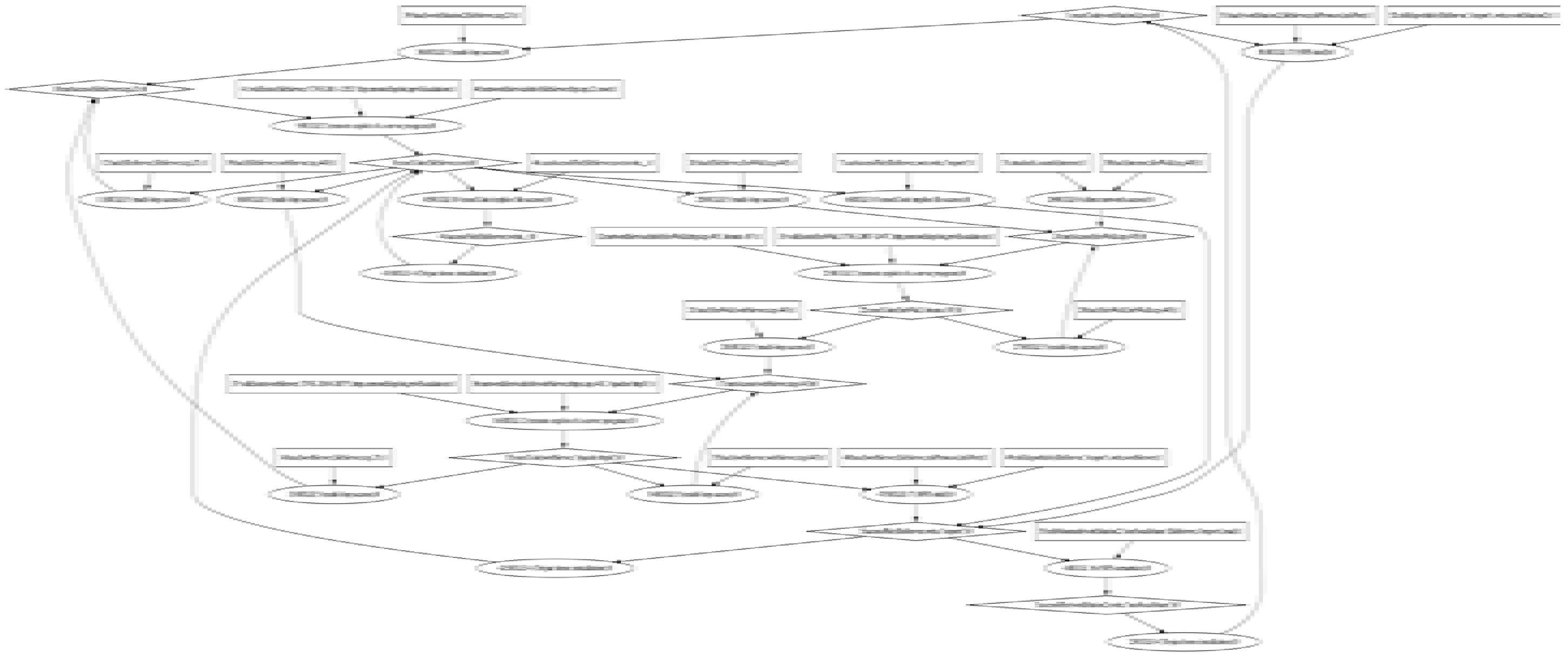
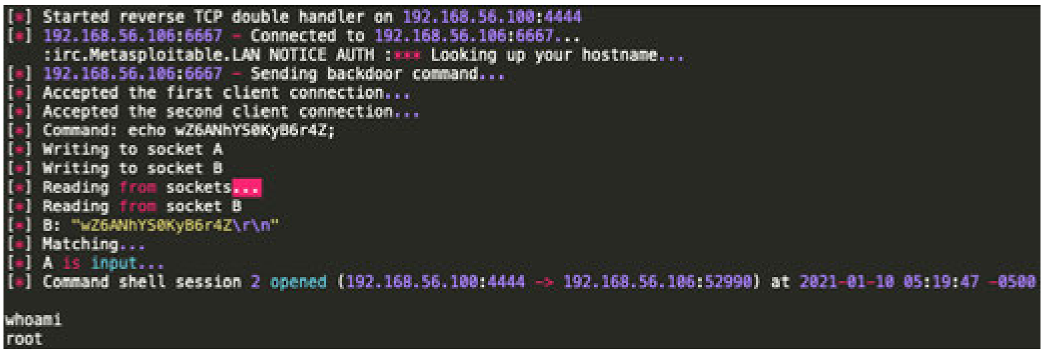


Figure 4.13: Attack tree for real network.

of the machine ‘Web server’, Fig. 4.14 shows the exploiting process of the machine ‘Web server’. After exploiting successfully we get a session which can be communicated with the next machine, the result of sending command “whoami”” is ‘root’ which shows that we gain the root privilege successfully. The attack process of ‘Firewall’ and ‘File server’ is basically the same as that of ‘Web server’.

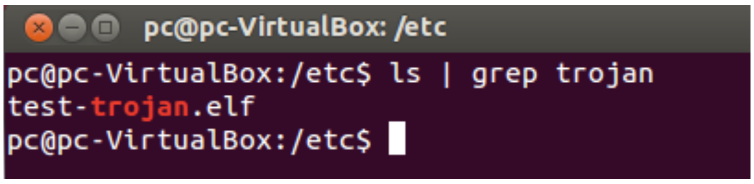


```
[*] Started reverse TCP double handler on 192.168.56.100:4444
[*] 192.168.56.106:6667 - Connected to 192.168.56.106:6667...
    :irc.Metasploitable.LAN NOTICE AUTH :*** Looking up your hostname...
[*] 192.168.56.106:6667 - Sending backdoor command...
[*] Accepted the first client connection...
[*] Accepted the second client connection...
[*] Command: echo wZ6ANhYS0KyB6r4Z;
[*] Writing to socket A
[*] Writing to socket B
[*] Reading from sockets...
[*] Reading from socket B
[*] B: "wZ6ANhYS0KyB6r4Z\r\n"
[*] Matching...
[*] A is input...
[*] Command shell session 2 opened (192.168.56.100:4444 -> 192.168.56.106:52990) at 2021-01-10 05:19:47 -0500

whoami
root
```

Figure 4.14: Exploiting process of Web server.

After gaining the root privileges on all three machines, a tunnel between ‘Internet’ and ‘Workstation2’ is established through port forward. The malicious Trojan will forward from ‘Internet’ to ‘Workstation2’. Fig. 4.15 shows the existed Trojan on ‘Workstation2’. When the Trojan is been executed, we can easily access the ‘Workstation2’ from ‘Internet’ to achieve the goal of penetration testing.



```
pc@pc-VirtualBox: /etc
pc@pc-VirtualBox:/etc$ ls | grep trojan
test-trojan.elf
pc@pc-VirtualBox:/etc$ 
```

Figure 4.15: Trojan file uploaded to Workstation2.

The overall attack process can be described as the following: The attacker first obtains the vulnerability information and attack path of the target host through the scanning function of Nmap, and then the information is input to the penetration tools, next the Metasploit is called through the RPC API to perform penetration testing on the host machine.

4.6 Discussion

We use different host and vulnerability information to simulate the experimental network that is evenly distributed among the target hosts. At the same time, the tool named ‘topology-generator’ is used to generate a large number of random network topology models to increase the diversity of training models. The purpose of this experiment is to demonstrate the feasibility of exploring and utilizing the framework in a variety of network environments. In the real environment, when a company’s network security protection is very good, the penetration testing team may often spend too much time trying to classify the security problems and do the penetration testing, and the final reward on investment is low. When the batch size is set to 16, this framework can provide the best attack path for the network. Even in the worst case when the value of GAMMA is too low or too high, the algorithm will converge in time and generate a feasible attack path.

The range of using this framework can be extended to many different areas. For example, we can use this framework to improve the company’s internal security protection. Testers can use this framework to easily evaluate the company’s network environment with relatively little experience. At the same time, this framework can also be used in the field of cybersecurity education, especially in penetration testing training activities. Trainees can try to do penetration testing in a simulated network environment. This will greatly reduce the threshold of organizing network education and training, because professional penetration testers are no longer needed. It will greatly improve the trainees’ network security protection ability.

Chapter 5

Conclusion and Future Work

5.1 Concluding Remarks

In this research, we utilized reinforcement learning (RL) techniques, in particular deep Q-learning network (DQN), to create an automated penetration testing framework. Firstly, we discussed the current situation of domain-related machine learning algorithms applied to automated penetration testing, and pointed out the necessity of using deep reinforcement learning to plan attack path discovery under automated penetration testing conditions. For this reason, we designed a DQN Decision Engine based on deep Q-learning network (DQN) to find the optimal attack path automatically. The engine adopted a three-layer neural network, and the learning parameters were carefully selected aiming for the best learning results.

Secondly, in order to generate enough network topology samples to improve the adaptability to different networks of DQN model, we integrated a topology generator tool into our framework to create random network topologies. Moreover, since the original matrix converted from the network topology is large, increasing the convergence time of the DQN model, we employed the DFS algorithm to simplify the matrix of attack path, thus improving the efficiency of the model learning.

Next, we employed several penetration tools to make it possible to used the DQN Decision Engine for penetration testing in a real network environment. Shodan was employed to collect network information for training the DQN model, Nmap was used to scan the network information, especially vulnerabilities of every machine, and RPC API was utilized to communicate with Metasploit to perform the real attack operations.

Finally, we conducted experiment that demonstrate our framework has a good performance both on the logical network and real network environment.

We used 1000 sample network topologies to evaluate the performance on the logical network, the accuracy of 'Topology 1' is 0.946 and the accuracy of 'Topology 2' is 0.918, and the average accuracy is 0.932. We also built a real network environment which is the same as 'Topology 2' to evaluate the performance on the real network. The framework exploited three different vulnerabilities to access the servers and copy the test Trojan to 'Workstation 2'. We hope that our research can promote the development of automated penetration testing, and also inspire other researchers.

5.2 Future Work

Our current research on automated penetration testing, especially on the attack path discovery algorithm, is only suitable for small-scale network scenarios due to its high computational complexity, and has some limitations on performing effective and fast attack path discovery in large-scale network scenarios. Therefore, it is important to do research on fast attack path discovery techniques in large-scale network scenarios. In the future, we will conduct additional research on how to quickly discover attack paths under large-scale network topology models.

Meanwhile, our research mainly focuses on single planning, and does not consider the effectiveness of the planned path. To overcome this problem, incorporating the generated feedback information during the process of penetration testing into the attack path discovery algorithm to dynamically adjust the attack path is a great way. Feedback information is an effective reflection of the attack load and host status information. Integrating information into the algorithm can effectively eliminate invalid attack loads and improve the effectiveness of the planned path. For example, the load cost is dynamically modified according to the feedback information, the cost of effective attack load behavior is reduced, the cost of invalid load behavior is increased, and the heuristic function is been guided to minimize the planning cost, thereby achieving the purpose of improving the effectiveness of the planned path.

In the end, we plan to improve the compatibility of our system with different penetration testing tools in the future. The current framework mainly use Nmap for the network scanning part, and Metasploit for the real attack part, but several other excellent tools exist. Therefore, we plan to integrate other network scanning tools, such as Nessus, and attack tools, such as Cobalt Strike, to AutoPentest-DRL in order to expand its applicability.

Bibliography

[1] W. Andrew, D. P. Newman, Penetration Testing and Network Defense, Cisco Press, 2005.

[2] B. Burns, E. Markham, C. Iezzoni, P. Biondi, M. Lynn, Security Power Tools, O'Reilly, 2007.

[3] J. P. McDermott, Attack net penetration testing, in: Proceedings of the 2000 workshop on New security paradigms, 2001, pp. 15–21.

[4] S. Jajodia, S. Noel, Topological vulnerability analysis: A powerful new approach for network attack prevention, detection, and response, Algorithms, Architectures and Information Systems Security (Indian Statistical Institute Platinum Jubilee Series) (2008) 285–305.

[5] J. Hoffmann, The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables, Journal of Artificial Intelligence Research 20 (2011) 291–341.

[6] Y. Stefinko, A. Piskozub, R. Banakh, Manual and automated penetration testing. benefits and drawbacks, in: 13th International Conference on Modern Problems of Radio Engineering, Telecommunications and Computer Science, 2016, pp. 488–491.

[7] F. Abu-Dabaseh, E. Alshammari, Automated penetration testing: An overview, Computer Science & Information Technology (2018) 123–127.

[8] B. Schneier, Attack trees - modeling security threats, Dr. Dobb's Journal 24 (1999) 21–29.

[9] J. L. Obes, C. Sarraute, G. Richarte, Attack planning in the real world, arXiv preprint arXiv:1306.4044 (2013).

[10] M. Boddy, J. Gohde, T. Haigh, S. Harp, Course of action generation for cyber security using classical planning, in: Proceedings of the Fifteenth International Conference on International Conference on Automated Planning and Scheduling, ICAPS'05, 2005, p. 12–21.

[11] J. L. Obes, C. Sarraute, G. G. Richarte, Attack planning in the real world, arXiv: Cryptography and Security (2013) 3–6.

[12] M. Yousefi, N. Mtetwa, Y. Zhang, H. Tianfield, A reinforcement learning approach for attack graph analysis, in: 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), 2018, pp. 212–217.

[13] G. Hinton, Y. LeCun, Y. Bengio, Deep learning, Nature 521 (7553) (2015) 436–444.

[14] R. S. Sutton, A. G. Barto, Reinforcement Learning: An Introduction, 2nd Edition, The MIT Press, 2018.

[15] Z. Zhou, Machine Learning, Tsinghua University Press, 2016.

[16] Y. Gao, S. Chen, X. Lu, Research on reinforcement learning technology: a review, Acta Automatica Sinica 30 (1) (2004) 86–100.

[17] M. C. Ghanem, T. M. Chen, Reinforcement learning for efficient network penetration testing, Information 11 (1) (2020) 6.

[18] J. Schwartz, H. Kurniawati, Autonomous penetration testing using reinforcement learning, arXiv preprint arXiv:1905.05965 (2019).

[19] L. P. Kaelbling, M. L. Littman, A. W. Moore, Reinforcement learning: a survey, Journal of Artificial Intelligence Research 4 (1996) 237–285.

[20] C. Li, L. Cao, Y. Zhang, X. Chen, Y. Zhou, Knowledge-based deep reinforcement learning: a review, System Engineering and Electronics 39 (11) (2017) 2603–2613.

[21] C. J. Watkins, P. Dayan, Q-learning, Machine learning (1992) 279–292.

[22] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing Atari with Deep Reinforcement Learning, in: NIPS Deep Learning Workshop 2013, 2013, pp. 201–220.

[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, Nature 518 (2015) 529–533.

[24] C. Watkins, Learning from delayed rewards, Ph.D. thesis, Cambridge University (1989).

[25] O. Anschel, N. Baram, N. Shimkin, Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning, in: International Conference on Machine Learning, PMLR, 2017, pp. 176–185.

[26] G. Xiang, Y. Cao, A study on detection-oriented attack classification and attack tree generating algorithm, Transactions of Beijing Institute of Technology 23 (3) (2003) 340–344.

[27] X. Ou, S. Govindavajhala, A. W. Appel, Mulval: A logic-based network security analyzer., in: USENIX security symposium, Vol. 8, Baltimore, MD, 2005, pp. 113–128.

[28] G. Kaur, N. Kaur, Penetration testing–reconnaissance with nmap tool., International Journal of Advanced Research in Computer Science 8 (3) (2017).

[29] M. Shah, S. Ahmed, K. Saeed, M. Junaid, H. Khan, et al., Penetration testing active reconnaissance phase–optimized port scanning with nmap tool, in: 2019 2nd International Conference on Computing, Mathematics and Engineering Technologies (iCoMET), IEEE, 2019, pp. 1–6.

[30] D. Maynor, Metasploit toolkit for penetration testing, exploit development, and vulnerability research, Elsevier, 2011.

[31] J. Meng, A. Li, The implementation of vulnerability scanning technique based on loading nessus on metasploit, Netinfo Security 12 (8) (2012) 185–186.

[32] R. Mudge, Network attack collaboration, sharing the shell, USENIX ;LOGIN: 36 (6) (2014) 27–28.

[33] O. Sheyner, J. Haines, S. Jha, R. Lippmann, J. M. Wing, Automated generation and analysis of attack graphs, in: Proceedings 2002 IEEE Symposium on Security and Privacy, IEEE, 2002, pp. 273–284.

[34] S. Khan, S. Parkinson, Towards automated vulnerability assessment, in: 11th International Scheduling and Planning Applications Workshop (SPARK), 2017, pp. 33–34.

[35] Z. Hu, R. Beuran, Y. Tan, Automated penetration testing using deep reinforcement learning, in: 2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), IEEE, 2020, pp. 2–10.

[36] B. Cheikes, D. A. Waltermire, K. Scarfone, Common platform enumeration: Naming specification version 2.3, National Institute of Standards and Technology (2011).

[37] M. Karlsson, The edit history of the national vulnerability database, Master's thesis, Swiss Federal Institute of Technology Zurich (2012).

[38] T. Grance, M. Stevens, M. Myers, Guide to selecting information technology security products, National Institute of Standards and Technology (2003).

List of Publications

Z. Hu, R. Beuran, Y. Tan, Automated Penetration Testing Using Deep Reinforcement Learning, in: 2020 IEEE European Symposium on Security and Privacy Workshops (Euro S&PW), IEEE, 2020, pp. 2–10.