2023 International Conference on Information Processing and Network Provisioning (ICIPNP) | 979-8-3503-6347-0/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICIPNP62754.2023.00100

Design Methodology for an Automated Penetration Testing System based on Improved Monte Carlo Tree Search

1st Yishuo Wang
National Computer System Engineering Research Institute of China
Beijing, China
wangyishuo0401@163.com

2nd Chaobin Huo*
National Computer System Engineering Research Institute of China
Beijing, China
huocb@ncse.com.cn

3rd Shaojie Wang
National Computer System Engineering Research Institute of China
Beijing, China
wangshj@ncse.com.cn

4th Minchao He
National Computer System Engineering Research Institute of China
Beijing, China
hemch@ncse.com.cn

5th Yiting Liu
National Computer System Engineering Research Institute of China
Beijing, China
liuyt@ncse.com.cn

***Abstract*—This paper presents a design methodology for an automated penetration testing system, drawing inspiration from the AlphaGo concept. In this paper, an improved Monte Carlo tree search method (MCTS) is used, which combines the traditional MCTS algorithm with the trained deep neural network to guide the node expansion, so as to achieve faster and better attack path determination. In the lateral movement process of penetration testing, the system uses the improved MCTS algorithm to select the next action in the search space according to the current environment state and known vulnerability information. Firstly, the DQN algorithm is used to train and learn in the environment, and then the trained deep Q-network is used to provide more reliable decision-making guidance in the search process, so as to speed up the speed of finding the optimal attack path. The experimental results show that the automated penetration testing system proposed in this paper has achieved significant improvement in the path planning task in the post-penetration testing stage.**

Keywords—Penetration testing, Monte Carlo tree search, Deep Q-network

I. INTRODUCTION

The automated penetration testing system emulates human penetration testing behavior by scanning and exploiting vulnerabilities within the target system. It assesses the system's vulnerability based on the level of difficulty associated with exploiting these vulnerabilities. Moreover, it possesses the capability to conduct repeated penetration tests from various entry points. While this technology holds great promise for the future, it is still in its nascent stage and faces several challenges that need to be addressed.

Indeed, automated penetration testing offers several distinct advantages. Firstly, it significantly reduces the overall testing time by leveraging automation. This efficiency is further enhanced as it enables simultaneous testing of multiple targets and continues testing without requiring human intervention, thereby streamlining the overall testing process. Furthermore, automated penetration testing provides better repeatability as the test execution is controlled and results can be recorded and reproduced. This feature is particularly beneficial for continuous integration and post-fix retesting, allowing for more reliable and consistent assessments. Additionally, automated penetration testing helps lower costs by reducing labor expenses and the need for extensive testing equipment, thereby enhancing the cost-effectiveness of the testing process.

Traditional penetration testing follows a structured framework comprising seven stages. The focus of this paper is primarily on the intranet penetration stage during the post-penetration testing phase. Specifically, upon obtaining permission for a designated host, penetration testers are required to utilize this host as a "pivot" for transferring traffic, enabling further exploration and exploitation within the network. In the subsequent stages of penetration testing, the focus shifts to conducting further tests on other host services within the internal network until all designated targets have been thoroughly evaluated.

II. RELATED WORK

A. Attack Tree

Attack tree model: In order to express the interconnection between attack actions and stages, Schneier initially introduced the concept of attack tree model[1]. Within the attack tree paradigm, each node denotes an attack action or an intermediary objective, with the primary root node symbolizing the ultimate target of the offensive operation. When the attack behavior associated with a child node is executed, it is considered to have successfully targeted the entity represented by its parent node.

MulVAL: MulVAL is an open source attack tree generation tool that can generate a complete attack tree based on the input network topology[2]. MulVAl uses Open Vulnerability and Assessment Language(OVAL) for input. OVAL uses XML format to describe vulnerabilities, patches, software versions and configuration information. It provides a standard, reusable framework that enables security tools to work better with each other.

The automated penetration testing system described in this paper employs MulVAL to generate a comprehensive attack tree based on all possible attack paths within the target network. Subsequently, a reinforcement learning algorithm is utilized to perform path planning on this attack tree.

B. Deep reinforcement learning

Reinforcement learning is a learning approach that involves mapping the state of an environment to actions, enabling an agent to maximize the rewards obtained during its interactions with the environment[3]. The ultimate goal is to develop a strategy that maximizes the long-term reward accumulation in the given environment.

979-8-3503-6347-0/23/$31.00 ©2023 IEEE
DOI 10.1109/ICIPNP62754.2023.00100

Deep Q Network (DQN) [4][5]simulates the Q-function using a neural network. To optimize the training process, DQN employs two identical neural networks: the target Q-network and the estimated Q-network. The target Q-network generates the target Q-values, while the estimated Q-network approximates the Q-values based on the current state.

C. Monte Carlo Tree Search and its Improvements

This paper draws inspiration from the AlphaGo system developed by DeepMind[6].

MCTS, is a simulation-based search algorithm commonly employed to tackle complex game problems[7]. The algorithm evaluates the state of the game by conducting simulations on the game tree, enabling it to determine the optimal action at each step based on the accumulated results from these simulations. Specifically, the MCTS algorithm consists of the following four steps: Selection, Expansion, Simulation and Backpropagation.

These four steps are called an iteration, as shown in the Fig. 1, and the algorithm returns the most optimal action in the search tree within a preset period of time or after reaching a preset number of searches.

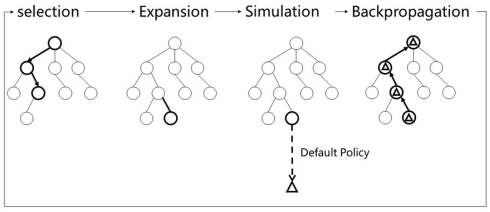


Fig. 1. One iteration of the Monte Carlo tree search

AlphaGo trained a deep neural network, referred to as a "Policy Network," which takes the current board state as input and produces a probability distribution over the available moves. The policy network learns to predict the optimal next move by analyzing vast amounts of historical data during training.

During the search process, AlphaGo feeds the current board state into the policy network to obtain a probability distribution for each possible move. Subsequently, the MCTS algorithm is employed to select the optimal move based on the generated probability distribution.

As the policy network has learned from extensive training data how to predict the most favorable move, AlphaGo can effectively and accurately identify the optimal next move through neural network-assisted search, significantly surpassing the capabilities of traditional random simulations.

D. Modeling of penetration test network model

DQN is a value-based reinforcement learning algorithm. It is crucial to realistically model the real network state into an environment that can be trained for reinforcement learning agents. Because there is a very powerful open source vulnerability exploitation framework metasploit framework (MSF), which integrates a large number of mature exploit programs, Therefore, this paper abstracts the action of Vulnerability exploitation, and uses the Common Vulnerability Scoring System (CVSS) to evaluate the risk degree of the vulnerability itself.

III. SYSTEM STRUCTURE

In this chapter introduces the structure of the proposed automated penetration testing system framework, the framework includes four parts, as shown in Fig. 2 shown below:

- Attack graph generation module: Generates an attack graph based on the information collected by the information collection module.
- Information collection module: Collect and analyze the information of the target system, scan the system for vulnerabilities, and obtain the network topology of the target system.
- Reinforcement learning path planning engine: training on the attack graph, to generate the optimal attack path, used to guide the exploit module work.
- Exploit Module: A set of tools for performing actions on real systems.

A. Information collection module

The function of the information collection module is to collect and analyze the information of the target system to obtain valuable information. At the same time, it is also necessary to collect the vulnerability information that has been published so as to better evaluate the security state of the target system. In this paper, Nessus is used for information detection and vulnerability scanning of the target system, because it can output the scanning results in OVAl format, which is convenient for MulVAl to generate the attack graph later.

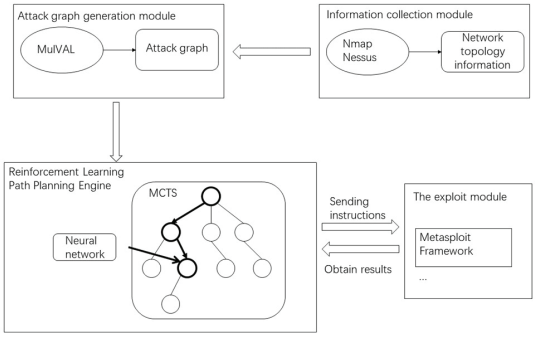


Fig. 2. Automated penetration testing system structure

In order to better evaluate the scanned vulnerabilities, we established a vulnerability database containing all CVE vulnerability data[8], including the CVE number of the vulnerability, the vulnerability type, the base score and the exploitable score in the CVSS score when storing the vulnerability, as shown in TABLE I.

TABLE I. VULNERABILITY DATA FORMAT

CVE Number	Type	BaseScore	ExpScore
CVE-2017-7160	Buffer Errors	6.8	8.6
CVE-2017-7161	Command Injection	6.8	8.6
CVE-2017-7174	Security Features	7.5	10.0

B. Attack graph generation module

The function of the attack graph generation module is to convert the attack path into a graphical attack graph model, which shows how the attacker attacks in the target system. The attack graph will also be used as input to a reinforcement learning path planning engine for training reinforcement learning agents.

Referring to the input format of MulVAl, define the host in the network topology, as shown in Fig. 3:

```
vulExists(webServer, 'CVE-2012-0053', https).
vulProperty('CVE-2012-0053', remoteExploit,
privEscalation).
networkServiceInfo(webServer, https, https, 80, 'Apache
httpd').
nfsExportInfo(webServer, '/export', _anyAccess, fireWall).
```

Fig. 3. The definition of host

Subsequently, the conversion of the attack tree into a transition matrix becomes imperative. Yousefi et al. have presented a technique for this transformation process in their research[9]. He transformed the attack graph generated by MulVAL into the form of a transition matrix, and the value on the coordinates (a,b) in the matrix represents the cvss score of exploiting the vulnerability when moving from node a to b. A value of -1 means that the two points are not reachable, and a value of 100 means that the final destination has been reached. However, it does not consider actions such as file access and code execution in penetration testing. Therefore, we refer to the methods used by Zhenguo Hu et al[10] and Rohit et al[11], firstly, the initial step involves the mapping of all nodes within the attack tree into matrix format, and the corresponding vulnerability scores are calculated to fill in the matrix. At the same time, some operations other than vulnerability exploitation were predefined to fill in the matrix with a specific cost. It also adjusts the final cost based on the services used by the system, such as ssh, ftp, and so on.

C. Reinforcement learning path planning engine

In the context of penetration testing, path planning is usually not about maximizing reward, but minimizing cost. Therefore, during the training of the agent, we can set its cost to a negative value and the neural network will still optimize to maximize this value. The attack cost of each vulnerability is defined as follows:

$$c_i = 10 - s_i \tag{1}$$

where s_i is the CVSS score of vulnerability i.

The input to the DQN model is the current state and the output is a probability distribution over each possible next action.In the context of the learning process, the DQN model agent assumes the role of the attacker, while the target environment is abstractly represented by a simplified attack matrix. Within this framework, the attacker has the capability to transition between nodes in the attack matrix until ultimately reaching the ultimate objective, namely, the target server.

When training the DQN model, we use the Prioritized Experience Replay (PER) mechanism to solve the problem that the more valuable experience cannot be better utilized due to random sampling during experience replay[12]. In this mechanism, each example is assigned a priority, which reflects its contribution to the learning of the agent. The priority is proportional to the TD (Temporal Difference) error of the sample, that is, the higher the prediction error, the higher the priority. During experience replay, samples are sampled according to their priority, and samples with higher priority are selected more frequently. The TD error is the difference between the current estimate and the estimate at the next time step.It is calculated as follows:

$$\delta = R + \gamma V(S') - V(S) \tag{2}$$

where δ represents the TD error, R represents the reward in the current state, γ represents the discount factor, $V(S')$ represents the value estimate in the next state, and $V(S)$ represents the value estimate in the current state.

When the DQN model is trained, the Q-network in it is used to guide the MCTS algorithm to select the penetration action. In the expansion stage of MCTS, the nodes predicted by the neural network are preferentially expanded. The specific process is shown in the pseudocode 1. Where s is the state, v is the node, s(v) is the state associated with node v,a (v) is the action that led to the state of v, N(v) is the number of times v has been visited, Q(v) is the reward for the simulation of v, and f(s,a) is a mapping from (state,action) to state.

Algorithm 1 Improved MCTS Algorithm

```
function IMPORVED MCTS(s0)
    create root node v0 with state s0
    while within computational budget do
        vl ← TREEPOLICY(v0)
        Δ ← DEFAULTPOLICY(s(vl))
        BACKUP(vl ,Δ)
    end while
    return a(BESTCHILD(v0,0))
end function
function TREEPOLIC(v)
    while v is nonterminal do
        if v not fully expanded then
            return EXPAND(v)
        else
            v ← BESTCHILD(v,1)
        end if
    end while
    return v
end function
function EXPAND(v)
    choose a ∈ untried action from A(s(v)) using
deep neural network prediction
    add a new child v' to v with s(v') = f(s(v), a)
and a(v') = a
    return v'
end function
function BESTCHILD(v, c)
    return max Q(v')/N(v') + c sqrt(2lnN(v)/N(v'))
end function
function BACKUP(v, Δ)
    while v is not null do
        N(v)←N(v)+1
        Q(v)←Q(V)+Δ(v,p)
        v←parent of v
    end while
```

```
end function
function DEFAULTPOLCY(s)
    while s is non-terminal do
        choose a ∈A(s) uniformly at random
        s ← f(s,a)
    end while
    return reward for state s
end function
```

D. Exploit Module

In order for an automated penetration testing system to attack a real system, it needs to interact with the actual network working environment, such as running commands, exploiting vulnerabilities, and so on. At this phase, the paper opts to employ established penetration testing tools, such as the Metasploit Framework[13].

IV. EXPERIMENTAL RESULTS

To substantiate the efficacy of the proposed approach, a network topology instance is employed as an illustrative example. The Reinforcement Learning path planning engine, as explicated earlier, is employed to ascertain the optimal attack path.

A. Experimental scenario

Fig. 4 depicts the network topology model employed in the experimental analysis conducted in this study. The configuration encompasses a web server,witch run for three docker container, a mail server, a file server, two subnets, a router, and a workstation. Specifically, the workstation is interconnected with the file server. Web server, mail server, Subnet1, Subnet2, file server are linked to each other through a router. The comprehensive device specifications are presented in TABLE II. .

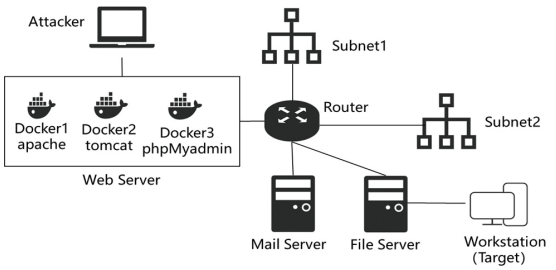


Fig. 4. Experimental scenario

TABLE II. EXPERIMENTAL SCENARIO HOSTS CONFIGURATION

Host	Vulnerability	Product/Service	Protocol
Mail Server	-	-	NFS RPC
Docker1	CVE-2002-0392	apache	http
Docker2	CVE-2017-12615	tomcat	http
Docker3	CVE-2018-12613	phpMyadmin	http
Web Server	CVE-2002-0392	docker	-
Subnet 1	CVE-2010-0483	windows-2000	http
Subnet 2	CVE-2010-0490	IE	http
File Server	CVE-2010-0812	windows-2003-server	NFS
Workstation	CVE-2010-0491	acrobat	NFS

The experimental scenario involves current popular production scenes and utilizes virtualized Docker containers. This necessitates performing docker escape operations during the penetration process in order to proceed to the post-penetration phase.

B. Attack Graph Generation

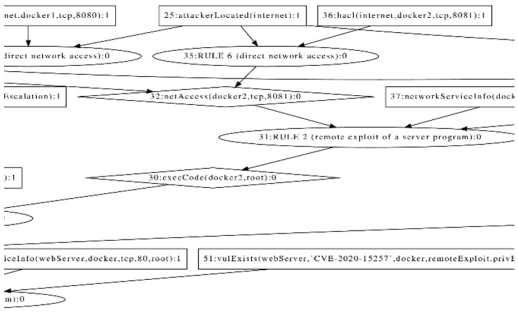


Fig. 5 Attack Graph(partial)

The network topology and configuration information described above is fed into MulVAL to generate the corresponding attack tree, as shown in Fig. 5.Considering the size of the attack graph, only a partial view of the attack graph is presented here. In the context of the attack graph, each node is annotated with a concise explanation of how to exploit a specific vulnerability. In this instance, the attacker initiates their intrusion through the Internet portal, denoted as node 25 in the top right corner of the diagram. The attacker's ultimate target is to execute code on the workstation, symbolized as node 1. It is assumed that the attacker can progress in the direction of the arrows until they reach the designated attack target.

In order to build the transition matrix required by the reinforcement learning algorithm, a reward score is assigned to each node, which in the penetration testing scenario represents the cost required for each step. The definition is as follows:

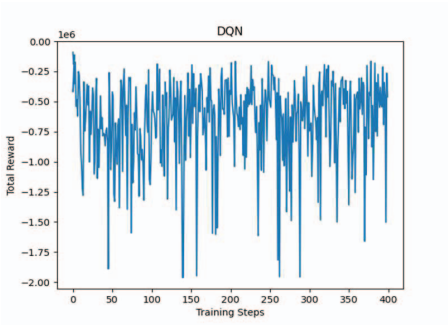
- The starting node (node 25 in this scenario) has a reward value of 0.01 and the goal node (node 1 in this scenario) has a reward score of -100.
- For every node involved in the exploitation of vulnerabilities, like the instance of exploiting CVE-2002-0392 at node 27, the reward score is determined using the formula referred to as 1.
- Regarding nodes where code execution or file access occurs, a standardized reward score of 0.3 is assigned for each, as these actions are considered equally significant within the penetration testing context.
- For all remaining nodes within the tree, a reward score of 10 is allocated. In the event that there is no viable path connecting two nodes, a reward score of 999 is assigned.

C. Reinforcement Learning training results

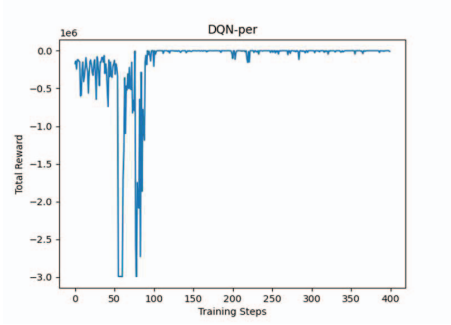
In this paper, a comparative analysis is conducted between the original Deep Q-Network (DQN) algorithm and DQN_{per}, which incorporates the priority experience replay mechanism. The outcomes are presented in Fig. 6, showcasing the

variations in total reward achieved by both algorithms in relation to the attack path during 400 training iterations within the experimental scenario.

Observations reveal that employing the original DQN algorithm in the experimental environment yields challenges in convergence within a limited number of training iterations, resulting in consistently negative and highly unstable rewards. However, with the integration of experience replay, the reward starts to exhibit an upward trend around the 80th training session, attains positive values by the 100th training session, and stabilizes at 39.7. It should be noted that the fluctuation in the reward curve arises due to the utilization of the ϵ-greedy strategy during agent training. This implies that even in later stages of training, random attempts may still be made, potentially leading to suboptimal attack paths. Hence, it is imperative to combine this algorithm with the MCTS algorithm for path planning, as the DQN algorithm alone cannot guarantee optimal outcomes.



(b) DQN Training Result



(b) DQNper Training Result

Fig. 6. Comparison of Deep Reinforcement Learning Training Results

The deep Q network trained by DQN is combined with the MCTS algorithm to guide its expansion operation and finally find the optimal path: $25 \rightarrow 46 \rightarrow 43 \rightarrow 42 \rightarrow 41 \rightarrow 39 \rightarrow 15 \rightarrow 14 \rightarrow 13 \rightarrow 10 \rightarrow 9 \rightarrow 8 \rightarrow 7 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1$.

The experiment demonstrates that in the context of virtualized container scenarios, the model successfully identifies attack paths, accomplishes escape operations, and compromises the target host in the post-penetration testing phase. Through the trained deep neural network guiding the MCTS algorithm, the system can quickly and accurately determine the optimal attack path, thereby improving the efficiency and success rate of penetration testing.

V. Conclusion

The automated penetration testing system, which is founded on the principles of AlphaGo as outlined in this paper, has achieved remarkable results in the path planning task in the post-penetration testing phase. By employing the MCTS algorithm guided by the trained deep neural network, the system can quickly and accurately determine the attack path, and improve the efficiency and success rate of penetration testing.

In addition to guiding the MCTS algorithm, the deep neural network can also be used to evaluate the overall security state of the system. Because the output of the neural network is the probability distribution of the next behavior based on the current state, it may take the second most probable behavior, the third most probable behavior, etc., simulate multiple attack paths, and rank the vulnerability of these paths. Rectify according to the urgency of the vulnerability. This can effectively evaluate the security of network systems and help enterprises find potential vulnerabilities and weak links. At the same time, this system can be used as an auxiliary tool for the training of network security engineers to help them learn and understand the methods and techniques of penetration testing, and improve the actual combat ability and response ability of engineers. The attack path of this system can be used as a reference for the red team attack behavior, helping enterprises to understand the attacker's thinking and methods, so as to improve and strengthen the network security defense line, and enhance the overall security.

References

[1] Schneier B. Attack trees[J]. Dr. Dobb's journal, 1999, 24(12): 21-29.

[2] Ou X, Govindavajhala S, Appel A W. MulVAL: A logic-based network security analyzer[C]//USENIX security symposium. 2005, 8: 113-128.

[3] Sutton R S, Barto A G. Reinforcement learning: An introduction[M]. MIT press, 2018.

[4] Mnih V, Kavukcuoglu K, Silver D, et al. Playing atari with deep reinforcement learning[J]. ar**v preprint ar**v:1312.5602, 2013.

[5] Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning[J]. nature, 2015, 518(7540): 529-533.

[6] Silver D, Huang A, Maddison C J, et al. Mastering the game of Go with deep neural networks and tree search[J]. nature, 2016, 529(7587): 484-489.

[7] Browne C B, Powley E, Whitehouse D, et al. A survey of monte carlo tree search methods[J]. IEEE Transactions on Computational Intelligence and AI in games, 2012, 4(1): 1-43.

[8] Karlsson M. The edit history of the national vulnerability database[J]. Master's thesis, 2012.

[9] Yousefi M, Mtetwa N, Zhang Y, et al. A reinforcement learning approach for attack graph analysis[C]//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). IEEE, 2018: 212-217.

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

[11] Gangupantulu R, Cody T, Park P, et al. Using cyber terrain in reinforcement learning for penetration testing[C]//2022 IEEE International Conference on Omni-layer Intelligent Systems (COINS). IEEE, 2022: 1-8.

[12] Schaul T, Quan J, Antonoglou I, et al. Prioritized experience replay[J]. ar**v preprint ar**v:1511.05952, 2015.

[13] Kennedy D, O'gorman J, Kearns D, et al. Metasploit: the penetration tester's guide[M]. No Starch Press, 2011.