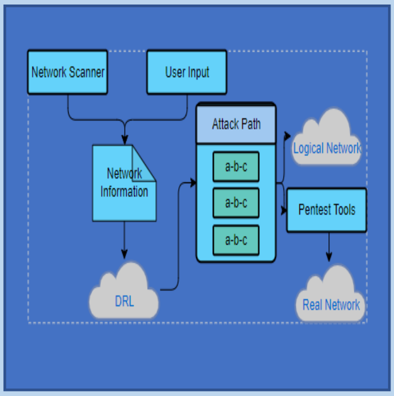
An-Najah University Journal for Research – A
Natural Sciences

Penetration Testing and Attack Automation Simulation: Deep Reinforcement Learning Approach

Ismael Jabr[1,2], Yanal Salman[1,3], Motasem Shqair[1,4] & Amjad Hawash[1,*]

Received: 11th Sep. 2023. Accepted 24th Mar. 2024, Published: 1th Feb. 2025, DOI: 10.35552/anujr.a.39.1.2231

Abstract: In this research, we propose a revolutionary deep reinforcement learning-based methodology for automated penetration testing. The suggested method uses a deep Q-learning network to develop attack sequences that effectively exploit weaknesses in a target system. The method is tested in a virtual environment, and the findings indicate that it can identify vulnerabilities that manual penetration testing is unable to. A variety of tools, including Deep Q-learning network, MulVAL, Nmap, VirtualBox, Docker, National Vulnerability Database (NVD), and Common Vulnerability Scoring System (CVSS), are used in this work. The suggested method significantly outperforms current automated penetration testing methods. Our proposed methodology can detect flaws that manual penetration testing misses and can be modified (in terms of penalty values) to adapt to the updates of the target system (network) changes. Additionally, it has the potential to greatly enhance penetration testing's effectiveness and efficiency and could contribute to the increased security of computer systems. Experimental tests conducted in this work reveal the effectiveness of DQN automated penetration testing by utilizing the most effective attack vectors in the attack automation process.



Keywords: Pentesters, Nmap, DQN, MulVAL, CVSS.

INTRODUCTION

The increasing reliance on Internet services has made network security a critical concern due to a surge in security incidents and cyber-attacks [1]. Issues include the abuse of user account privileges, unidentified network assets, unpatched vulnerabilities, and insufficient IT security management. To enhance network security, actions such as packet monitoring, timely patch upgrades, strong authentication, encryption, network segmentation, and access controls are essential [2, 3]. Penetration testing, a popular solution, involves ethical simulated cyber-attacks to assess network security levels. Ethical hackers, or pentesters, use tools and undergo extensive training for effective testing. While manual penetration testing is complex and time-consuming, it follows stages like planning, information gathering, and exploiting vulnerabilities [4]. Recently, attempts have been made to integrate Artificial Intelligence (AI) techniques into areas such as Vulnerability Assessment, Exploitation, and Reporting in network security [5].

Attack modeling is a process where models of potential attack trajectories are created to model security threats on the topology of a particular system. It plays a vital role in understanding the relationship between each potential attack method to be used in penetration testing. There are three famous attack models: Attack Tree, Attack Graph, and Planning Domain Definition Language (PDDL) [6]. To ensure that automated penetration testing is as effective as human-led penetration testing, attack graphs are a valuable tool. It enables pentesters to better understand the relationship between each attack method. The use of Reinforcement Learning (RL) is to analyze attack graphs, which uses Q-learning to find attack paths. However, it has a problem with the fact that the workspace and sample area are very small. Meanwhile, the use of Deep Reinforcement Learning (DRL) is a better and more appropriate option especially when the samples or scenarios are small since it combines deep learning with enhanced learning and adopts a trial-and-error approach to find the optimal solution [7].

In this paper, we present a framework for simulating attack automation and penetration testing, created and carried out to determine the optimum attack path for a specific topology. Our contribution to this work is related to maximizing the benefit of penetration testing which is considered faster and more accurate than manual testing considering the changes in each network and does not rely on a fixed sample of networks [22]. Our effort in this work is to emphasize the importance of automating penetration testing and emerge the benefits of this automation to assure the problems of manual testing and how the automation can list all possible attack paths to maximize the amount of security for a given network.

Our research contribution entails conducting a simulation study for penetration testing, which can be summarized by the following steps:

1. Utilizing virtualization and containerization techniques to construct testing network topologies for the penetration testing process.

1 Department of Information & Computer Science, Networks & Information Security Program, Faculty of Engineering & Information Technology, An-Najah National University, Nablus, Palestine,
* Corresponding author email: amjd@najah.edu
2E-mail: ismael1@protonmail.com, 3 E-mail: Ynal.salman@gmail.com, 4 E-mail: chrismotasem11@gmail.com,

2. Employing Nmap to gather network data and utilizing NVD as a comprehensive repository of known vulnerabilities worldwide.
3. Constructing an attack graph to visualize potential attack paths using the MulVAL tool.
4. Applying Deep Q-Learning Network (DQN) to analyze the attack matrix and identify the optimal attack path based on CVSS v3 and CVSS v2 scores.
5. Executing automated breakthroughs following the paths we found.
6. Evaluating the success or failure of the hacking process to show the effectiveness of DQN for automated penetration testing.

The proposed approach is highly comprehensive in detecting vulnerabilities across any environment. It scans for the most common vulnerabilities, including both old and new ones. The approach also includes automated steps to check for the presence of exploitable vulnerabilities. Furthermore, the Deep Q-learning algorithm learns from all information about the topology, information assets, and vulnerabilities to optimize the detection process for the optimal path and other possible paths that the threat actor may use to compromise the environment.

The approach relies on the following steps:

1. Scanning for vulnerabilities based on topology and asset information, which may not detect zero-day vulnerabilities or new vulnerabilities without updating the tool containing the scanner and exploit.
2. The Q-learning algorithm learns the optimal path and other possible paths that threat actors may use to compromise the environment based on the information from the previous step.
3. Verifying the existence and exploitability of vulnerabilities in the scanner results by performing automated attacks on all paths identified in the previous step to ensure that they are real and exploitable (no false positives).

The rest of this paper is organized as follows: Previous works are reviewed in Section II. Section III discusses the methodology we followed to implement the work while Section IV is related to the conducted experimental test. Finally, Section V concludes this paper.

Related Works

Network penetration testing is very important and it is considered a crucial frequent technique for most of today’s establishments. It has a lot of noticeable advantages: increases the personnel awareness regards networks’ break-ins from malicious entities, provides solutions that help in detecting and preventing attackers, measures the number of risks in networks, and decreases the number of errors that might take place [8].

Although penetration testing can be executed internally or externally, there are different techniques used in penetration testing. Social engineering, web applications, physical penetration, network services, and client-side, and wireless security tests are all a set of examples [9].

According to the research described in [10], penetration testing may be automated by employing rule trees, and each chain of rule trees contains a complete assault process. The authors of the article suggest that the security assessment process should adhere to universal standards by leveraging the results of penetration testing. The proposed method, according to the authors, can increase security assessment accuracy and efficacy by continuously expanding rule trees.

The work presented in [23] is related to reviewing the empirical works of penetration testing and their findings. The authors of the work identified several potential research challenges and opportunities, such as scalability and the need for real-time identification of exploitable vulnerabilities.

According to the work presented in [24], the authors of the work propose a method aimed at automating the vulnerability discovery and mitigation process typically performed by Red Hat hackers. They exploit a toolchain of several well-known tools and they evaluated the proposed method by exploiting the Metaesploitable Linux distro, showing that the proposed method can automatically mitigate vulnerabilities afflicting six widespread services.

The work presented in [25] focuses on creating an automated penetration testing framework for smart home-based IoT devices. It examines common vulnerabilities and identifies necessary tools for detection. The framework, written in Python 3.6, is then used to test the devices individually for known vulnerabilities. The Tp-Link Smart Bulb and Tp-Link Smart Camera were found to be the most vulnerable, while the Google Home Mini was the most secure. The framework doesn't require technical expertise and can be used by the public, improving IoT security and ensuring a safe future for smart homes.

The work of [11] is related to presenting details of an example to illustrate how the authors of the work specify and analyze network attack models. They take sample models as input to automate the generation of attack graphs to analyze the vulnerability of some tested systems. Although the authors have previously published work related to some algorithms of penetration testing, this work is related to illustrative examples and the usage of some toolkits.

In terms of carrying out automated penetration testing that is based on deep reinforcement learning, the work presented in [12] is very similar to ours. To provide guided learning for attack training, the authors of the work constructed a testing framework to be utilized as a component of cybersecurity training activities. The framework was used to indicate potential attack techniques. However, our contribution is different from this work in using API[1] (National Vulnerability Database) to get CVSS (Common Vulnerability Scoring System) version 3.0 values, CVSS v3.0 is more accurate than CVSS v2.0 when evaluating the paths using DRL 2[2] and to apply automated attack to get the best path to follow in the process of searching for vulnerabilities. We used Virtualization technology, which is a feature that allows you to create additional virtual environments that run on physical devices through specialized programs to run more than one operating system on the same device at the same time [13].

In general, our work presented in this paper differs from the mentioned works (the manual and the automated) in more than one point. The work lacks penalties that steer the searching process for the best attacking path, and some of these works are related to some previously prepared examples and lack the dynamicity that is an important feature of our work.

Background and Methodology

The methods, tools, and strategies employed in this study to replicate automated attack and penetration testing are described in this section. However, our work here is based on the work and

[1] https://nvd.nist.gov/
[2] If not found, code gets to version 2.0 values.

equations presented in [12] to automate the penetration testing that is the core of our work.

A. Virtualization Technology

The use of virtualization technology is considered the best approach for performing penetration testing research due to several reasons, with ethical considerations being one of the primary factors. Virtualization allows researchers to create isolated and controlled environments for conducting security assessments without risking harm to real-world systems or violating ethical boundaries. By running penetration tests within virtual machines, researchers can simulate various attack scenarios, assess vulnerabilities, and test security controls in a controlled setting.

By using a specialized program to run multiple operating systems on the same hardware device simultaneously, virtualization technology gives users the ability to build extra virtual environments that function on their actual equipment. An example of this technology is hypervisor technology [13]. It allows multiple operating systems to work simultaneously on a host. It is divided into two types in terms of interacting with the machine hardware. One type interacts directly and the other through the hosting operating system. We relied on virtualization technology to create a real network topology scenario to test the techniques we used to simulate penetration testing and complete the attack on these virtual systems. We used Docker and Virtual-Box to combine the two previously explained types of Hypervisor.

B. Common Vulnerability Scoring System

As a free and open-source industry standard for judging the seriousness of computer system vulnerabilities, the Common Vulnerability Scoring System (CVSS) [14]. The Forum for Incident Response and Security Teams (FIRST), a non-profit organization with more than 500 members worldwide, owns and runs CVSS. Its goal is to support incident response teams for computer security all over the world. The severity of the information security vulnerability is represented numerically (0–10) by this system. This system gives a numerical rating of the seriousness of the information security vulnerability (0–10). The National Vulnerability Database (NVD) is the American government's repository for vulnerability management data that adheres to standards and is represented by the Security Content Automation Protocol (SCAP). Nearly all known vulnerabilities are given CVSS scores by NVD. Both CVSS v2.0 and v3.X standards are supported (We rely on CVSS v3.0 values and if they are not available for the vulnerability we bring CVSS v2.0 values through the API). By revealing the privileges necessary to exploit the vulnerability and the ability of an attacker to spread across systems (or "scope") after doing so, CVSS v3.0 was created to solve some of the flaws of its predecessor, v2. The ratings between CVSS v2.0 and CVSS v3.0 are displayed in Table I.

Table (1): NVD Vulnerability Severity Ratings.

Severity	Base Score Range	
	CVSS V 2.0	CVSS V 4.0
None	-	0.0
Low	0.0-3.9	0.1-3.9
Medium	4.0-6.9	4.0-6.9
High	7.0-10.0	7.0-8.9
Critical	-	9.0-10.0

We used NVD in our work because it provides CVSS scores for nearly all known vulnerabilities. We created a dynamic data set using the Nist API associated with NVD so that it gives us the degree of CVSS v3.1 for the vulnerability. We configured the work to use v2.0 in the case v3.0 is not available [15].

C. Attack Graph

A cyber-attack graph shows all potential attack routes against a cybersecurity system and shows the status after a successful breach has been carried out by an attacker [16].

1. Attack Graph Model: The attack graph is a brief representation of all possible avenues, paths, and possible cases of attacks against the topology of the absolute network. It can be considered as a way of expressing the relationship between attack behavior and attack steps to document the attack process. Each node in the graph can represent a host, vulnerability, or network device [17].
2. MulVAL: MulVAL: MulVAL is an open-source program used to generate a real attack graph for a given network architecture for network security risk assessment. MulVAL relies on Datalog as an object modeling language for analysis. MulVAL takes network topology and security vulnerabilities extracted from the network via Nmap as inputs by expressing its outputs in Datalog and feeding them to our MulVAL reasoning engine. Once inserted into MulVAL, it can analyze networks with thousands of devices in seconds. In our framework, we used MulVAL to find all possible paths to attack our network topology. It builds a matrix according to potential attack paths in the graph using a depth search first (DFS) algorithm to make it more suitable for use with the Deep Reinforcement Learning (DRL) algorithm [18].

D. Reinforcement Learning

Reinforcement learning is an area of machine learning that aims to learn the optimal policy and is responsible for increasing or decreasing the expected cumulative reward based on the reward or punishment of actions [19]. In general, reinforcement learning is divided into the following aspects:

1. The algorithm's or AI's agent is in charge of choosing a course of action.
2. The environment: consists of various circumstances that the agent might encounter.
3. The reward signal: a signal that the environment sends back based on the situation at hand.
4. The agent is moved from one state to another by each action.
5. The policy: a mapping of the agent's behavior from states to actions.

The projected cumulative reward for each condition under the current policy is provided by the value function, also referred to as the Q function, in the Reinforcement Learning (RL) approach. In the sense that it provides an answer to the query: What will be my anticipated reward if I start in state I and follow my policy? The expected cumulative reward is discounted by some factor $\lambda \in (0,1)$[1]. The conflict between the urge to pick the best-known choice and the need to attempt something new to uncover alternative possibilities that could be even better is one of the dilemmas inherent in the RL problem setting. Exploitation refers to selecting the most well-known activity, whereas exploration refers to selecting a novel action. Usually, a little probability of exploration is incorporated into the method to address this. The strategy might be, for instance, to choose the best course of action with a probability of 0.95 and then randomly choose another course of action with a probability of 0.5 (if the probability distribution is uniform across all remaining courses of action, it is given by $0.5/(n-1)$, where n is the number of

[1] A typical value of λ is 0.9.

states). The interaction between the agent and environment is seen in Figure 1.

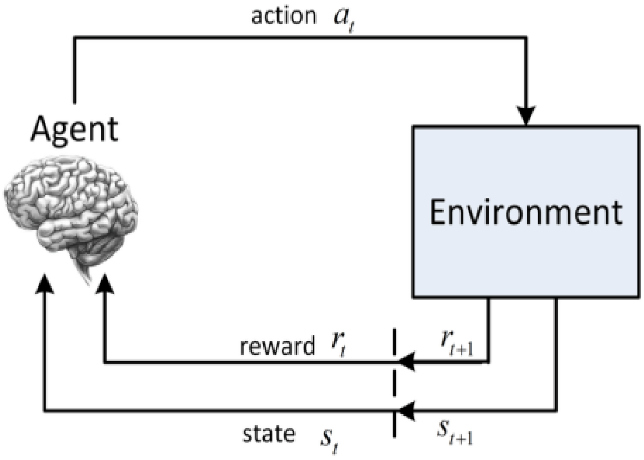


Figure (1): Reinforcement Learning Process where a set of rewards are given to the agent according to his activity with the system.

At each time step t, the agent looks at the surroundings to obtain the state S_t and then performs the action At. Markov Decision Processes (MDP) can be used to represent such a process. The environment generates the new state $St + 1$ and the reward Rt depending on At.

A MDP consists of four tuples, (S, A, P, R), where S is a set of states known as the state space; A represents the action set known as the action space; $P\ (s' \mid s, a)$ is the likelihood that taking action in state s at time t will result in state s' at time $t + 1$; and $R(s, a)$ is the immediate reward (or expected immediate reward). The strategy's objective is to select a course of action at each time step that maximizes future cumulative rewards, so the future cumulative rewards of the current state may be used to assess the quality of that state [12].

To determine the reward at any given time, reinforcement learning introduces an equation known as a reward function t:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma r_{t+1+k} \qquad (1)$$

where γ is the discount coefficient. Because the reward value is more uncertain the further away from the current state, it is commonly used to account for this uncertainty. However, this function accumulates and maximizes the amounts of future rewards to compute the quality of the current state. Reinforcement learning also introduces equation 2 that is called a value function to represent the value of a given state, i.e. the expectation of future cumulative rewards for that state [12]. In other words, the value of $V^{\pi}(s)$ is computed by the expectation of the future accumulative value applied in equation 1:

$$V^{\pi}(s) = E[\sum_{k=0}^{\infty} \gamma r_{t+1+k} \mid s_t = s] \qquad (2)$$

Equation 3, known as the action-state value function, is also provided by reinforcement learning and is used to express future cumulative rewards that are conditioned by both the state and the action, also for the accumulative future reward for a given state applied in equation 1:

$$Q^{\pi}(s, a) = E[\sum_{k=0}^{\infty} \gamma r_{t+1+k} \mid s_t = s, a_t = a] \qquad (3)$$

E. Deep Reinforcement Learning (DRL)

Deep Reinforcement Learning is a sub-field of machine learning that combines two parts: Reinforcement Learning and Deep Learning [19]. Deep Learning is evolved from a machine learning method known as a perceptron or multilayer perceptron, which has gained increasing interest in recent years because of its effectiveness in a variety of sectors ranging from computer vision to signal processing, medical diagnosis, and self-driving automobiles. Deep learning encompasses a lot more than a standard artificial neural network. However, neural networks and perceptron networks used in machine learning have a significant impact.

DRL is a general learning technique where the agent receives status information from the environment, chooses the appropriate actions based on his or her strategies, changes the state of the environment, and then receives a reward that, depending on the new environment's state, determines the effectiveness of the agent's actions. DRL algorithms can process very big inputs and decide what steps should be taken to advance the target. When DRL is applied to penetration testing, the agent takes on the role of a pentester and selects the best (most effective) route to maximize reward. Value-based operations, search strategies based on strategy, and model-based procedures make up the three fundamental categories of DRL algorithms.

We applied the Deep Q-Learning Network (DQN) algorithm, a crucial subset of the value-based DRL algorithm, as the DRL algorithm [19]. DQN combines the Convolution Neural Network (CNN) with the Q-Learning algorithm in traditional enhanced learning to create the new DQN model. DQN model input is a simplified matrix coming out of MulVAL. The use of DQN was to experience all possible paths of attack and to extract the best paths from them (optimal paths) through the continuous training of the DQN model based on reward for paths where the path with the highest reward is the optimal path and then that follows and so on.

The reward associated with exploiting each vulnerability used in the DQN model is determined by a vulnerability score that we developed based on components of the Common Vulnerability Scoring System (CVSS):

$$Score_{vul} = baseScore \times \frac{exploitablityScore}{10} \qquad (4)$$

The $baseScore$ in CVSS measures the severity of the vulnerability, whereas the $exploitablityScore$ shows the ease with which the vulnerability may be exploited. In order to balance the importance of the base score when considering the viability of exploiting a specific vulnerability, we used the exploitability score, which has a maximum value of 10. In Figure 2, we introduce the DQN training method, which improves on the standard Q-learning technique to solve difficulties like instability in the non-linear network's representation function. DQN, for example, processes transfer samples using experience replay [12]. The transfer samples obtained by the agent interacting with the environment at each time step t are stored in the replay buffer unit. During the training process, a small batch of transfer samples is chosen at random and the network parameter θ is updated using the Stochastic Gradient Descent (SGD) algorithm.

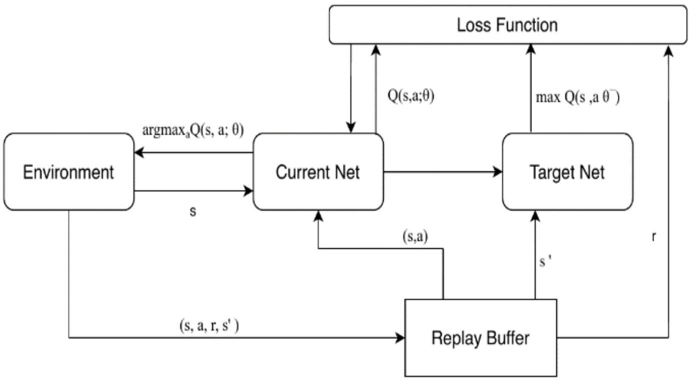


Figure (2): DQN training process where replies of agents interact with the testing environment.

DQN also alters the method of calculating the Q value. $Q\ (s, a \mid \theta i)$ is the output of the current value network in DQN, and it is used to assess the value function of the current state action. The output of the target value network is $Q\ (s, a \mid \Theta_i^-\ )$, and the goal Q value is given by equation 5, where Y_i is generally adopted as the target of maximizing Q value [12].

$$Y_i = r + \gamma\ max_{a'} Q(s', a' \mid \Theta_i^-) \qquad (5)$$

The current value network's parameter θ is updated in real-time. The parameters of the current value network are replicated into the target value network after every N round of iteration. The network parameters are then updated by reducing the mean square error between the current Q value and the target Q value. The error function is as follows:

$$L(\theta_i) = E_{s,a,r,s'}[(Y_i - Q(s,,a, \mid \theta_i))^2] \qquad (6)$$

and the gradients are computed as follows:

$$\nabla_{\theta_i} L(\theta_i) = E_{s,a,r,s'}[(Y_i - Q(s,a \mid \theta_i))\nabla_{\theta_i} Q(s,a \mid \theta_i)] \qquad (7)$$

Equation 6 computes the average squared difference between the estimated values and the actual one since it measures the quality of estimation process. The amounts of gradients in computing the amounts of errors are given by equation 7 in order to estimate the change of error values during the whole process of finding the best path of ethical attack.

F. Attack Automation

The attack automation method, which makes up the majority of the automated penetration testing procedure, is at the center of our efforts. Therefore, for our automated penetration testing framework to be used to conduct attacks against real systems, it must be able to communicate with real network environments by issuing commands and finding security flaws [20].

Instead of creating our tools, we chose to use a well-known framework in the penetration testing community: Metasploit Framework. It includes manual brute forcing, manual exploitation, third-party import, and a command line interface. This free version of the Metasploit project also comes with a Ruby compiler since that is the language used to create this version of Metasploit [21]. We wrote a bash script to automate attacks and view the final result.

Our work is built on exploiting the DQN trainer model's output to give commands to penetration tools, which will carry out the steps on our target systems. We worked on fixing threshold values to focus on the dangerous vulnerabilities and neglecting the weak ones by conducting the attack process to ensure their authenticity. This leads to a quick report that includes the serious vulnerabilities and submits it to the responsible party to work on closing/fixing them as soon as possible. As for the rest of the other gaps, another report will be made containing tips and suggestions to raise the level of security, for example, adding an anti-virus, sandboxes, firewall, … etc. Figure 3 concludes the implemented steps in our proposed methodology. Please be notified that our proposed methodology works now for real networks and not logical ones. We included the logical network in the figure just to mention that our methodology works for both logical and real.

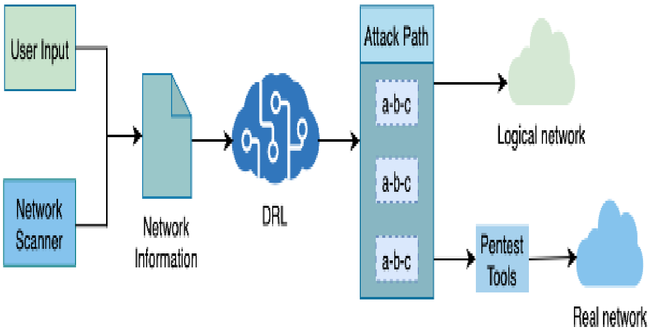


Figure (3): The whole proposed methodology steps.

Experimental Test

To perform actual penetration testing and assess the framework's suitability under actual conditions, we built a virtual machine environment.

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

- The initial point of entry for the attacker into the Web Server node, is through the Internet node, utilizing the HTTP and HTTPS protocols.

- The File Transfer Protocol (FTP) establishes a connection between the file server and web server nodes (The reason we chose the FTP protocol is that its attack scores are the highest in the CVSS file which means they are one the most important attacks that have to be considered carefully).
- The File Transfer Protocol (FTP) establishes a connection between the workstation and file server nodes.
- The Server Message Block (SMB) protocol establishes a connection between the workstation and web server nodes.

Virtual Machine Configuration and Tooling Setup in Research Environment

- Attacker VM: Utilized VirtualBox for installing the operating system and Docker for managing tool dependencies.
- Victim 0 VM: Deployed VirtualBox to install the operating system and Docker for setting up the vulnerable service.
- Victim 1 VM: Employed VirtualBox for installing the operating system.
- Victim 2 VM: Utilized VirtualBox for installing the operating system.

This list provides information about the virtual machines used in the research setup, highlighting the specific virtualization tools (VirtualBox) utilized for operating system installations. It also mentions the use of Docker for managing tool dependencies and setting up vulnerable services in the Attacker and Victim 0 VMs.

The executed penetration testing steps are shown as follows:

1. Nmap is used to find out each machine's vulnerability in a network setting.
2. By integrating the results of Nmap scanning with the configuration file's knowledge about the network architecture, MulVAL creates an attack graph.
3. The DQN Decision Engine receives an attack tree that has been converted into a matrix.
4. In order to carry out an attack using Metasploit, DQN Decision Engine computes and selects the best attack path before sending it to a bash script.

To construct the transfer matrix required by the DQN algorithm, each node must be assigned a reward score, which is done as follows:

- The start node reward value (node 26 in our example) is 0.01, and the goal node reward score (node 1 in our example) is 100.
- As the reward score for each node that exploits a vulnerability, we use the Scorevul value defined in equation 4
- We define a reward score of 1.5 for each node that executes code or accesses files, as such actions are critical during the penetration testing process (target node 1 is excluded from this rule).
- For any other node in the tree, we assign a score of 0, and if there is no path between two nodes, we assign a score of -1.

Figure 4 shows the procedures of the framework (scanning hosts using Nmap, building an attack graph using MulVAL, invoking CVSS data from API to create a matrix, and computing optimal attack paths using DQN). The bottom of the image contains a list of possible attack paths each of which indicates the list of nodes involved in the path.

Table II reflects the values computed by the DQN algorithm where the first entry of the ordered pairs represents the current node number and the second represents the node number to be attacked next. Table II shows the transfer matrix required by the DQN algorithm. The transfer matrix is a table that maps from one state to another. Each path should be considered in the process of penetration testing taking into account the reward values for each path.

The DQN algorithm uses the transfer matrix to learn how to generate attack sequences that successfully exploit vulnerabilities in a target system. The DQN algorithm starts in a random state, and it then tries to move to a state with a higher reward value. The DQN algorithm learns to move to states with higher reward values by trial and error. The red path in Table II represents the attack sequences with the highest reward values. This path is the most among the others with the higher probability of being attacked and should be considered in the penetration testing.

To evaluate our procedure in terms of effectiveness concerning discovering the list of paths that have the most probability of being attacked, we compared our approach with manual testing by 5 users. We provided the users with the necessary information about the network under test and asked them to write down the possible attack paths. Moreover, to test the speed of our approach compared with the manual one, we computed the time spent by the users to discover the possible attack paths. By comparing the results of our approach depicted in Figure 4 and Table II, we found that the users were able to discover (on average) 5 possible paths within 2 hours. However, our approach takes around 1 minute to discover the list of paths depicted in Figure 4 and Table II including those discovered by the users.

CONCLUSION

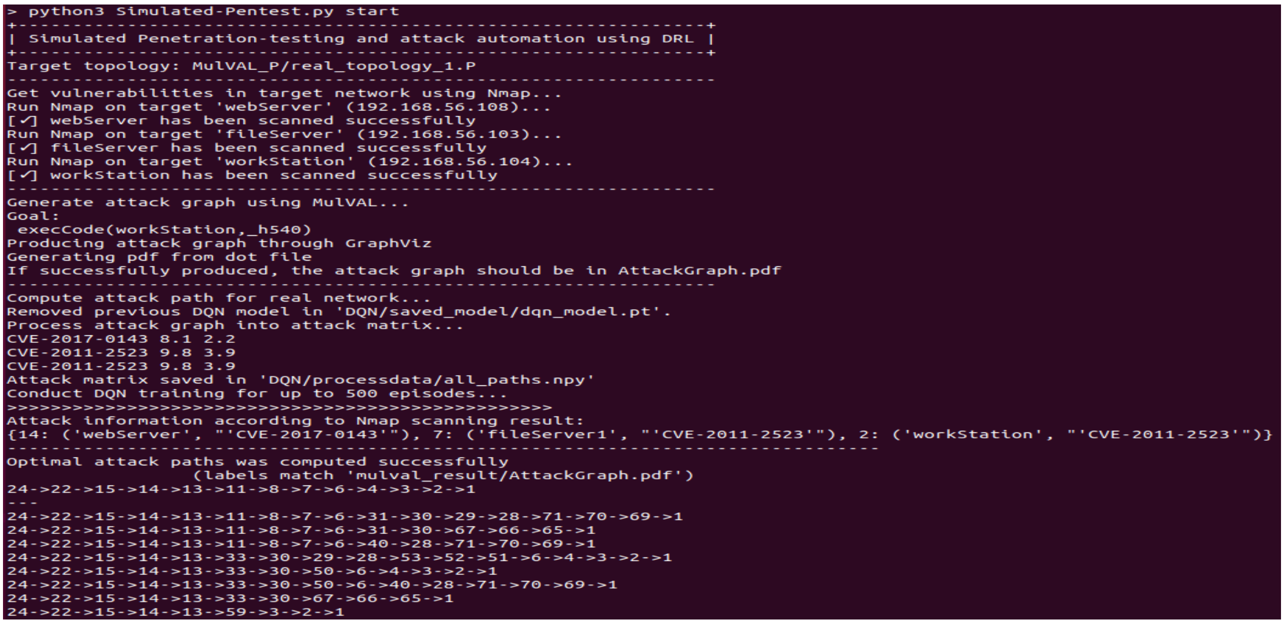
In this paper, we presented a Deep Q-Learning Network-based system for simulating penetration testing (DQN). Our method collected actual host and vulnerability data by utilizing API (NVD) to merge the Nmap scanner with all vulnerability information. The attack information for each of the training scenarios was subsequently generated using the attack graph methodology. The DQN model determined the most likely attack path for a specific network scenario based on reward ratings allocated to each node, primarily relying on CVSS score information. The effectiveness of DQN automated penetration testing was demonstrated by utilizing the most effective attack vectors in the attack automation process.

Moreover, complementarity with more Penetration testing tools that include Penetration testing for Network, Web, and Phone applications (Android and iOS) as well as Penetration testing for IoT and ICS/SCADA Devices…etc.

There are a lot of tools that can be used (including free and paid) such as Nessus and Cobalt Strike, as well as the possibility of using tools such as Vuls.io that scan apps and packages on the operating system for any vulnerabilities in the versions installed on the device.

Our future work will focus on improving attack automation to encompass all CVEs, exploring faster methods to identify attack paths in large-scale network topology models, and integrating two machine learning models for the MITRE attack framework and predicting zero-day vulnerabilities.

Forthcoming research endeavors will prioritize enhancing attack automation capabilities to encompass the entirety of CVEs. Additionally, efforts will be directed toward investigating expeditious techniques to discern attack paths within extensive network topology models. Furthermore, there will be a concerted focus on integrating two distinct machine learning models, one about the MITRE attack framework and the other concerning the prediction of zero-day vulnerabilities. These future undertakings aim to advance the field of cybersecurity by bolstering the efficacy of attack strategies and fortifying preemptive measures against emerging threats.



```
> python3 Simulated-Pentest.py start
+---------------------------------------------------------------+
| Simulated Penetration-testing and attack automation using DRL |
+---------------------------------------------------------------+
Target topology: MulVAL_P/real_topology_1.P
-----------------------------------------------------------------
Get vulnerabilities in target network using Nmap...
Run Nmap on target 'webServer' (192.168.56.108)...
[✓] webServer has been scanned successfully
Run Nmap on target 'fileServer' (192.168.56.103)...
[✓] fileServer has been scanned successfully
Run Nmap on target 'workStation' (192.168.56.104)...
[✓] workStation has been scanned successfully
-----------------------------------------------------------------
Generate attack graph using MulVAL...
Goal:
 execCode(workStation,_h540)
Producing attack graph through GraphViz
Generating pdf from dot file
If successfully produced, the attack graph should be in AttackGraph.pdf
-----------------------------------------------------------------
Compute attack path for real network...
Removed previous DQN model in 'DQN/saved_model/dqn_model.pt'.
Process attack graph into attack matrix...
CVE-2017-0143 8.1 2.2
CVE-2011-2523 9.8 3.9
CVE-2011-2523 9.8 3.9
Attack matrix saved in 'DQN/processdata/all_paths.npy'
Conduct DQN training for up to 500 episodes...
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
Attack information according to Nmap scanning result:
{14: ('webServer', "'CVE-2017-0143'"), 7: ('fileServer1', "'CVE-2011-2523'"), 2: ('workStation', "'CVE-2011-2523'")}
--------------------------------------------------------------------------------
Optimal attack paths was computed successfully
                 (labels match 'mulval_result/AttackGraph.pdf')
24->22->15->14->13->11->8->7->6->4->3->2->1
---
24->22->15->14->13->11->8->7->6->31->30->29->28->71->70->69->1
24->22->15->14->13->11->8->7->6->31->30->67->66->65->1
24->22->15->14->13->11->8->7->6->40->28->71->70->69->1
24->22->15->14->13->33->30->29->28->53->52->51->6->4->3->2->1
24->22->15->14->13->33->30->50->6->4->3->2->1
24->22->15->14->13->33->30->50->6->40->28->71->70->69->1
24->22->15->14->13->33->30->67->66->65->1
24->22->15->14->13->59->3->2->1
```

Figure (4): Framework Procedures Output. A list of possible attack paths appears at the bottom of the image.

Table (2): Transfer matrix required by DQN Algorithm where the first number represents the node number and the second is for the reward value.

24,22	22,15	15,14	14,13	13,33	33,30	30,29	29,28	28,71	71,70	70,69	69,1				
24,22	22,15	15,14	14,13	13,33	33,30	30,29	29,28	28, 53	53, 52	52, 51	51, 6	6, 4	4, 3	3, 2	2,1
24,22	22,15	15,14	14,13	13,33	33,30	30, 50	50, 6	6, 40	40, 28	28, 71	71, 70	70, 69	69, 1		
24,22	22,15	15,14	14,13	13,33	33,30	30, 50	50,6	6, 4	4, 3	3, 2	2, 1				
24,22	22,15	15,14	14,13	13,33	33,30	30, 67	67,66	66, 65	65, 1						
24,22	22,15	15,14	14,13	13,11	11,8	8,7	7,6	6, 31	31, 30	30, 29	29, 28	28, 71	71, 70	70, 69	69, 1
24,22	22,15	15,14	14,13	13,11	11,8	8,7	7,6	6, 31	31, 30	30, 67	67, 66	66, 65	65, 1		
24,22	22,15	15,14	14,13	13,11	11,8	8,7	7,6	6, 40	40, 28	28, 71	71, 70	70, 69	69, 1		
24,22	22,15	15,14	14,13	13,11	11,8	8,7	7,6	6, 4	4, 3	3, 2	2, 1				
24,22	22,15	15,14	14,13	13,59	59,3	3,2	2,1								

DISCLOSURE STATEMENT

- Ethics approval and consent to participate: Not Applicable
- Consent for publication: All authors listed have read and approved the final version of the manuscript and consent to its publication.
- Availability of data and materials: The raw data required to reproduce these findings are available in the body and illustrations of this manuscript.
- Author's contribution: The authors confirm their contribution to the paper as follows: study conception and design: Amjad Hawash, Ismael Jabr, theoretical modeling: Ismael Jabr, Moatasem Shqair; preparing testing cases, Yanal Salman. draft manuscript preparation: Amjad Hawash, Ismail Jabr. All authors reviewed the results and approved the final version of the manuscript.
- Conflicts of interest: The authors declare that there is no conflict of interest regarding the publication of this article

REFERENCES

1] Jang-Jaccard J, Nepal S. A survey of emerging threats in cybersecurity. J Comput Syst Sci. 2014;80(5):973-993.

2] Kumari L, Debbarma S, Shyam R. Security problems in campus network and its solutions. Int J Adv Eng Appl. 2011 Jan;Volume-1:98-101.

3] Bacudio A, Yuan X, Chu B, Jones M. An overview of penetration testing. Int J Netw Secur Its Appl. 2011 Nov;3:19-38.

4] Bertoglio D, Zorzo A. Overview and open issues on penetration test. J Braz Comput Soc. 2017 Dec;23.

5] Sarraute C. Using ai techniques to improve pentesting automation. 2010 Apr.

6] Moore A, Ellison R, Linger R. Attack modeling for information security and survivability. 2001 Jun.

7] Arulkumaran K, Deisenroth M, Brundage M, Bharath A. A brief survey of deep reinforcement learning. IEEE Signal Process Mag. 2017 Aug;34.

8] Shebli HMZA, Beheshti BD. A study on penetration testing process and tools. In: 2018 IEEE Long Island Systems, Applications and Technology Conference (LISAT); 2018. pp. 1–7.

9] Reza SS, Hasan W, Reza SS, Chakraborty S. A comparative overview on penetration testing. In: Proc. of The Fourth Intl. Conf. On Advances in Computing, Electronics and Electrical Technology–CEET; 2015. pp. 25–28.

10] Zhao J, Shang W, Wan M, Zeng P. Penetration testing automation assessment method based on rule tree. In: 2015 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER); IEEE; 2015. pp. 1829–1833.

11] Sheyner O, Wing J. Tools for generating and analyzing attack graphs. In: International symposium on formal methods for components and objects. Springer; 2003. pp. 344–371.

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

13] Hay B, Dodge R, Nance K. Using virtualization to create and deploy computer security lab exercises. In: IFIP International Information Security Conference. Springer; 2008. pp. 621–635.

[14] Mell P, Scarfone K, Romanosky S. Common vulnerability scoring system. IEEE Secur Privacy. 2006;4(6):85–89.

15] Franklin J, Wergin C, Booth H, et al. Cvss implementation guidance. National Institute of Standards and Technology, NISTIR-7946. 2014.

16] Jin W, Li Y, Xu H, Wang Y, Ji S, Aggarwal C, Tang J. Adversarial attacks and defenses on graphs: A review, a tool and empirical studies. 2020.

17] Liu C, Singhal A, Wijesekera D. Using attack graphs in forensic examinations. In: 2012 Seventh International Conference on Availability, Reliability and Security. IEEE; 2012. pp. 596–603.

18] Ou X, Govindavajhala S, Appel AW, et al. Mulval: A logic-based network security analyzer. In: USENIX security symposium. vol. 8. Baltimore, MD; 2005. pp. 113–128.

19] Arulkumaran K, Deisenroth MP, Brundage M, Bharath AA. A brief survey of deep reinforcement learning. arXiv preprint arXiv:1708.05866. 2017.

20] Enoch SY, Huang Z, Moon CY, Lee D, Ahn MK, Kim DS. Harmer: Cyber-attacks automation and evaluation. IEEE Access. 2020;8:129397–129414.

21] Kennedy D, O'gorman J, Kearns D, Aharoni M. Metasploit: the penetration tester's guide. No Starch Press; 2011.

22] Stefinko Y, Piskozub A, Banakh R. Manual and automated penetration testing. Benefits and drawbacks. Modern tendency. 2016. pp. 488-491. 10.1109/TCSET.2016.7452095.

23] McKinnel DR, Dargahi T, Dehghantanha A, Choo KKR. A systematic literature review and meta-analysis on artificial intelligence in penetration testing and vulnerability assessment. Comput Electr Eng. 2019;75:175-188. doi:10.1016/j.compeleceng.2019.02.022.

24] Filiol E, Mercaldo F, Santone A. A Method for Automatic Penetration Testing and Mitigation: A Red Hat Approach. Procedia Comput Sci. 2021;192:2039-2046. doi:10.1016/j.procs.2021.08.210.

25] Akhilesh R, Bills O, Chilamkurti N, Chowdhury MJM. Automated Penetration Testing Framework for Smart-Home-Based IoT Devices. Future Internet. 2022;14:276. doi:10.3390/fi14100276.