**Reinforcement Learning Approaches for Penetration Testing**

Table of Contents

[Table of Contents 1](#_Toc134000766)

[List of Figures 2](#_Toc134000767)

[List of Tables 3](#_Toc134000768)

[Selection of Reward in Reinforcement Learning for Auto PenetrationTesting 4](#_Toc134000769)

[I. Introduction 4](#_Toc134000770)

[Background 4](#_Toc134000771)

[Aim of Research 6](#_Toc134000772)

[Structure of Paper 6](#_Toc134000773)

[II. Approach 6](#_Toc134000774)

[III. Experiments / Testing / Measurements / Data 16](#_Toc134000775)

[IV. Analysis and Discussion of Results 18](#_Toc134000776)

[V. Conclusions 18](#_Toc134000777)

[VI. Future Work 19](#_Toc134000778)

[Appendix A: Literature Review 20](#_Toc134000779)

[References 40](#_Toc134000780)

List of Figures

Figure 2-1, Structure of penetration testing based on RL in NAS 10

Figure 2-2, Example of network model 11

Figure 2-3, Examples of machine definitions 11

Figure 2-4, Examples of exploitable services on a network 12

Figure 2-5, Example of topology definitions 12

Figure 2-6, Example of firewall configuration in the network 12

Figure 2-7, Initial graph of tiny Scenario 13

Figure 2-8, Initial graph of medium Scenario 13

Figure 2-9, Training Cycle of Reinforcement Learning 13

Figure 2-10, Determination of Q-value in tabular method 14

Figure 2-11, Determination of Q-value in DQN 14

Figure 2-12, Initial state of Tiny Scenario 15

Figure 2-13, Final states DQN agent given in NAS for tiny scenario 15

Figure 2-14, Final graph of the tiny scenario for the DQN agent given in NAS 16

Figure 2-15, Initial states of medium scenario 16

Figure 2-16, actions for last steps 16

Figure 2-17, Final state of the netthe work in Experiment 1 16

Figure 2-18, Final graph of Experiment 1 17

Figure 2-19, Final state of the network in Experiment 2 17

Figure 2-20, Fnetworktwork network in Experiment 2 17

Figure 2-21, Final state of netwthe ork the Experiment 3 20

Figure 2-22, Final graph network in Experiment 3 20

Figure 2-23, Final state of the work in Experiment 4 21

Figure 2-24, Final state of the network in Experiment 4 21

List of Tables

Table 2-1, Network information for Tiny and Medium Scenarios 15

Table 2-2, Information of RL Environment for Tiny and Medium Scenarios 15

Table 2-3, Parameters used in training of DQN agents 15

Table 2-4, Total steps and Reward of Experiment 2 and [1] 18 Table 2-5, Evaluation of Value for actions in the original reward function 18

Table 2-6, Evaluation of Cost for actiActionsthe original reward function 18

Table 2-7, Evaluation of Value for actiActionsthe new reward function 19

Table 2-8, Results of 10 experiments in Experiment 3 20

Table 2-9, Results of 10 experiments in Experiment 4 21

Table 2-10, Comparison of total steps and rewards 21

Selection of Reward in Reinforcement Learning for Auto PenetrationTesting

**Abstract**: Penetration testing has a main role in identifying the security of the network and becomes increasingly important as network interconnections increase rapidly. Penetration testing needs network experts who have full knowledge and rich experiences in network attack and security, however, it becomes more and more difficult to find such experts. Recently, as AI has experienced rapid growth in all parts of science and technology, auto penetration testing methods based on AI have been developed. Especially, many researchers are interested in reinforcement learning-based penetration testing. There are many papers and packages for automated penetration testing with reinforcement learning. However, most of them concentrate on the limitations of the reinforcement learning model for penetration testing.  To get optimal performance from reinforcement learning in penetration testing, it is very important to set the reward function correctly. In this paper, we develop a deep reinforcement learning-based penetration testing method using an efficient reward function.

*Keywords*—Cyber Security, Network Attack, Penetration test, Reinforcement learning, Reward,

# Introduction

## Background

In recent years, computer networks have been deployed with the latest security checks & controls to protect against external computer threats. However, the number of computer threats has increased with greater complexity, as well as the sophistication and frequency of those threats. It is important to deploy cyber security for safeguarding computer systems and important individual and government data [3, 4, 12, 14, 21]. One of the major practices being done nowadays is to do penetration testing to evaluate the current security of the system and recommend major upgrades or controls that are required. Penetration testing is an established mechanism to check the security of digital systems, varying from an individual application to a network of digital communications. It doesn’t include only websites or software applications; it also includes hardware and network setups. In penetration testing, an authorised attack is performed to find the security vulnerabilities that could be used by a hacker to overcome to exploit the system.

Penetrating testing is approved as an effective method to find the security vulnerabilities of the system as it is a simulation of what would be performed by an attacker. However, the effectiveness of this method comes with one drawback: it requires high skills and time to perform such penetration testing.

After the introduction of the European Data Protection Regulation, penetration testing has become a mandatory and crucial component for evaluating cyber security threats to systems. As the size and complexity of digital systems increase, it impacts the cost and requirements of performing such tests, and therefore there is an increasing demand for such security testing and quality assurance professionals among the big security testing firms across the globe. [2] Penetration testing is a multiple-stage process that requires highly professional and skilled resources because of the complexity of medium and large networks [3].

It is important to mention here that penetration testing and security functional testing are two different terms and perform their functions differently. Security functional testing can describe the system’s security-correct behaviour, while penetration testing is used to calculate the level of difficulty one needs to penetrate an organisation’s security controls without authorization and access to its information.

In penetration testing, an unauthorised user will attack the system with either manual or automated tools for assessing the security controls and in possible circumstances where they can be disabled. Through penetration tests, security professionals can address the security layers and possible vulnerability points and highlight the vulnerabilities of low, medium, and high severity. Businesses and their operations get high benefits from quality testing tools like penetration testing. For a business organisation, penetration testing is mandatory because it prevents the theft of financial data, complies with industry security requirements, and provides reliable information to both stakeholders and customers [4].

Penetrating testing is an effective method to evaluate the strength of existing security products and the major upgrades or actions required for such resources. Systems’ penetration testing provides the "existence of an issue" and a valid case for the organisation’s management to invest in overcoming it. [5].

One of the leading information technology (IT) and networking hardware and services firms, Cisco Systems, estimated in 2015 that there were 1 million unfilled system security jobs in the world. Currently, several checking tools have been developed that help to perform penetration testing and improve the security of systems. The Metasploit framework is such a popular tool that it has been in development since 2003 and is now available for Penetration testing.[8]. Metasploit includes a library of known security threats to perform for your system penetration testing. Also, it includes many useful tools to scan the infogatheredathering on a targetToolshe tools like Metasploit helps a Penetration Tester to perform testing at a high level rather than analyzing and finding the vulnerabilities at low levels with his manually defined checklists. The Penetration testing tools help the Testers to work quickly with a high level of reliability, while it creates a friendly and easy-to-use assessment method for entry-level testers. ​[9]. While such Testing Tools have been beneficial to the Cyber security testing industry, the problem lies in the availability of professional trainers to perform such operations. Also, the methods of manually testing the security of systems become a burden in the easing complexity of the system’s growth. A new trending approach for performing system testing with the complexity of system structures and growth is to apply such tests from Artificial Intelligence (AI) to automate the testing process and get quick outcomes, which are not possible in manual testing with the tools. The reliability and efficiency of testing have one approach where Artificial Intelligence is the technique for the process automation and it has taken an original concept of “Concept Graph” where an already designed computer network is utilized to launch a Test Attack and it shows the current threats and vulnerabilities in the existing system [10]​. To learn the possible ways in which a system hacker can enter the system through a security breach can be learned through Attack Graphs. Such Attack Graphs need a sound knowledge of the system to use them and find the exploits that are different or unrealistic from the attacker’s point of view or such attacks being performed in the real world. Further, it needs to construct a manual attack graph for each system being tested.

The Partially Observable Markov Decision Process is another approach that is used for similar systems’ testing. In the Partially Observable Markov Decision Process, the attack on a system modelled retain an assumption attack exploits isse partially added in the testing process and then perform his exploits and assuming the simulation process that such, while it models and writes the threats in its configuration during this simulation process. ​[11], [12]. Due to the computing mechanism of POMDPs, this approach can be performed on an individual machine but doesn’t scale perfectly in a big environment [12]​. More research is required to design methods that should automate pen testing for handling the threats and exploits possible in a system with their computational properties. A possible solution to overcome such problems is to use MDB the approach. In this approach, the configuration of the host computer status is ignored and the possibility of each possible attack is tested. [13]​.

The MDP method is computationally more feasible as compared to the POMDPs approach because it doesn’t require the understanding and knowledge of hosting and network configurations. The MDP approach also requires previous knowledge of the system because it understands each target computing machine as the same instead of defining possible targets for each machine and it requires success possibilities for every performed action. It affects the accuracy of selecting the right actions for every host because this method doesn’t utilize the knowledge of host and network configuration. Another similar approach that addresses ingress the problems without knowing the host and network properties is Reinforcement Learning [14]. The state representations are the set of actions that determine the actions achievements by a Reinforcement Learning Agent and these are performed and reward functions for the RL Agent. By communicating with its environment, the RL Agent learns about the actions’ policy. Reinforcement Learning gained high popularity when used to beat the agents of the World GO Champion. As compared to other testing procedures, the Reinforcement Learning has not been commonly adopted but in robotic tasks, the RL has been successfully adopted [15]​.

In cases where the detailed model of the environment is not known, a general approach is used by the Reinforcement Learning model. It can define the tasks in a reward function by selecting the actions which should be performed while Automated Penetrating Testing. The problem lies with the ever-changing state of exploits and threats in a network environment and with ongoing changes in the software and updates, It becomes difficult to manage an accurate and update ongoing for executing any of such actions.

There are certain challenges while adopting the RL Model because it required a large volume of data to understand and learn a policy of performing the best actions and methods for Automated System Testing. The RL Agent is trained by the data gathered through the simulations. Recently, it is important to know that RL-based penetration methods get more and more attractive.

The Number of packages for auto penetration testing using deep reinforcement learning attracted . The Mulval is a tool used to analyse network security vulnerabilities and it improves the visualization by deeply trimming the logic graph.

NAS is a Python package that is given in the format of Oan penAI gym Environment [1].

It is additionally planned to model network pen-testing at a better level of abstraction in order to permit fast prototyping and testing of algorithms. The network model defines the organization, connection and configuration of machines on the network and is defined by the tuple subnetworks, topology, hosts, services and firewalls. This model seeks to abstract away those aspects of a real world network that aren’t needed when creating autonomous agents, such as specific types of machine connections and the location of switches and routers. The objective of this abstraction is to keep the simulator as simple as possible while still working at the level that the agent is expected to work at, allowing it to decide which scans or exploits to use against a certain system and in what order. This basic network model is also employed to keep it as generic and scalable as possible.

[1] and [6] developed a deep Reinforcement learning algorithm based on NAS.

They get the result for small networks.

However, the number of states and actions will be increased rapidly when the number of hosts are increasing.

In Table 1, the states and number of actions are given

This produces serious problems to apply the RL method to penetration testing since training can’t be converged if the numbers of states are too big. [50]

To make the training converge with high numbers of states, it will need high computation and time.

Preceding papers mainly concentrate on how to apply RL methods to penetration testing. To apply the RL should find a method to reduce the time that training is converged.

## Aim of Research

In this paper, we are going to check the preceding Deep RL-based penetration testing methods and find the solution to apply the Deep RL method to big networks.

Firstly, we analysed the environment and RL algorithm in the preceding methods and find the problems.

Secondly, we should propose a new method to solve the problems.

To do it, updating for Reward will be concentrated.

How to set Rewards for actions are important elements to decide the performance of training and test of RL however preceding papers don’t concentrate on this.

In this paper, we are going to propose a new deep RL-based auto penetration testing method and compare the performance s.

## Structure of Paper

This paper constitutes 5 chapters.

Section 1 shows an introduction to the paper.

Section 2 shows the approach for this paper.

Subsection 2.1 describes the NAS environment, and tests and analyses the DQN agent on NAS.

Subsection 2.2 describes the updating DQN agent of NAS based on the results of subsection 2.1.

Section 3 represents the experiments for the proposed algorithm and shows the results.

Section 4 analyses the results of experiments and shows the performance of a new algorithm.

Section 5 contains the conclusions for this paper.

The Appendix contains tables, Figures and programs used in this paper.

# Approach

As described in the introduction, there exist some auto pen-testing methods using deep Reinforcement learning and also several packages.

Here, we develop DQN-based auto pen-testing methods using NAS which is a Python package to generate an environment for pen-testing. NAS generates an Environment so we can easily use deep learning python packages TensorFlow, Pytorch and openAI which are widely used to.

This section has two sub-sections.

The First part is to test the deep Q Reinforcement Learning-based Auto penetration testing method on tiny and big networks and analyse their performance.

We use the DQN agent which is given in [] and get the sources of it on [https://github.com/Jjschwartz/NetworkAttackSimulator](https://github.com/Jjschwartz)​.

The Second part is to update the penetration testing based on the analysis for the first part.

1. **Analysis of DQN agent-based penetration testing**

In this sub-section, we tested DQN-based penetration testing methods on two scenarios and analysed the performance.

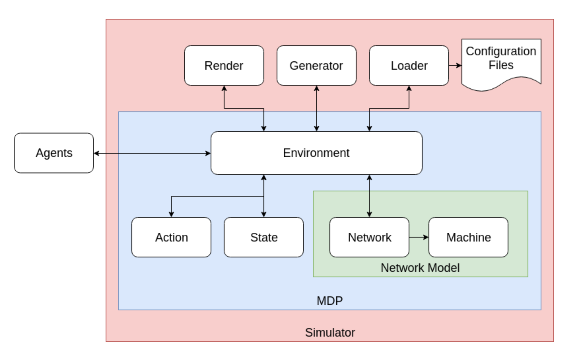


Figure 2-1. Structure of penetration testing based on RL in NAS.

Figure 2-1 shows the structure of penetration testing based on RL in NAS. As we can see from the Figure, penetration testing is constituted of two main parts, ie. agents and environment. The Agents receive the network information from the environment and attack the network based on it.

The Environment is given by NAS and constitutes several parts. The Network model is used to simulate a network environment and has two parts, ie. n network such as information of subnetworks, links between subnetworks and so on. The Machine simulates the host with os, service and process. The action consists of a scan action and a deterministic or non-deterministic exploit for each service and machine on the network. State is the set that contains the state of each machine on the network.

Loader is to load scenario file which has network information for subnetworks and hosts.

The Generator creates an openAI gym environment from a scenario file or benchmark scenario name.

Render is a library to show the state of the environment using Figures and shells. Detailed descriptions for NAS environments are given in the following subsections. [50]

* 1. **Environment**

1. ***Network Model***

In Network Model, the Organization’s network including server machines, workstations, firewalls, networks & Subnetworks, configuration and connectivity of the machines are defined by a tuple. In Figure 2-2, an Example of a Network Model is including installed firewall security, 16 machines and five subnetworks with a firewall is shown. In this network model, the number of services to run has been defined at the machine level so it can operate any number of the services. Further details about it have been mentioned in the below paragraphs. [34]



Figure 2-2. Example of the network model.

This model presents a real-time network model where details are not required in terms of the locations of routers, switches and machines. The major reason to use such a simulation model is that it is a simple and easy-to-learn presentation of how the agent will work in a simulation and which exploits or scans should be performed at specific machines connected in this network. Also, this model determines in which order the scans and exploits on the machines and the network. While adopting this practice in high-fidelity networks, the details of which ports to use for communication are handled by the implementation of the application in that environment. If the exploit works in a specified scenario, it is launched while knowing about all the details on the lower level of this exploit testing​[18]. In this network model for Exploit testing, the defined details for performing the actions are used so that it shall be easily scaled to the network map and to keep it as a standard method.

* **Machines**

In a Network Model, the machines are the most important building block. In NAS, the Machines refers to any type of device which is available on a network and which can be exploited through communicating the attacks in the network. Every machine while available on network has a unique Machine ID and Subnet ID along with its configuration. The Machines are defined with different values while performing an attack as a machine with high value is the one that the owner wants to protect or the Attacker wants to exploit to the low-value machine which is not important in terms of information available in it.

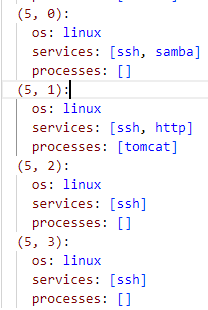


Figure 2-3. Examples of machine definitions

In **Figure 2-3**, the examples of Machines definition are given. The services running on every machine in a network can be communicated with other machines on the same network or other network but with the firewall or subnet giving permission to access it. It is not necessary that each machine on a network should have the same configuration and it depends on services available for all of the machines. The different machines available on a network have different purposes to serve and therefore their configuration varies, e.g. Server Machines, Workstations, File storage servers and web servers. Since the hackers are interested in exploiting the services running on those machines, the vulnerability of each machine differs and the level of security needed too.

* **Services**

After machines, the second major component included in a network model are the services running on the machines, and their level of security as, as well as their configuration,e different as defined above. The services include any type of software that the applications are running on the machines and communicating over the network. They may be compared to software that is communicating with software and devices available on the port. Services are vulnerability points in a machine network, and attackers may know about specific exploits that can hack the machine by accessing its services' communication in NAS.

In real-world scenarios, the networking protocols only keep track of services that are vulnerable or important to attacks and ignore those that may not be vulnerable. An agent’s job is to find the services running on each machine and apply the exploits they are vulnerable to. In the above reasoning for the NAS, we assume that one action can exploit each service. In Network Map, each service has been defined with a unique ID, the cost of applying the threat, and a possible percentage that the tester will be successful in applying the threat.

**Figure 2-3** shows how an attacker has certain exploits for the HTTP, SSH, and FTP services in a network, while Figure **2-4** shows the cost of exploits, the success probability of each exploit on these services and a set of services that can be exploited. Each service doesn’t need to have an actual environment name but rather a unique ID, as shown in the Figure. It is so that NAS shall generate IDs for the services with any number of services and machines in a defined test environment. However, the service names are used while performing the same configuration in real-world scenarios instead of defining IDs for the services. The services’ names (e.g. Samba Version 4.1.0) are used in a real environment for patching purposes and to specify the vulnerability of that version of the application.

* **Sub-Networks**

The Sub-Network is an important component in a network map. There are multiple subnets or subnetworks in every network.

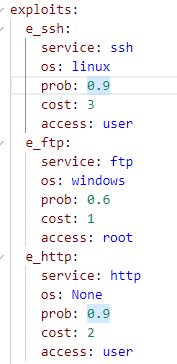


Figure 2-4. Examples of exploitable services on a network

The machines in a network are connected on a smaller network that is called a subnet and it may include a group of machines for communicating with each other in a larger network. The machines on a subnet have their own Subnet Address which is mentioned in its network address (e.g. 3 in (3,0)) is a subnet. It is a simple method to define the IP Addresses which have a 32-bit subnet mask or 32-bit string for defining the machine address, subnet or network. In NAS Network Model, we use a simple System as it is a single network working as compared to the use of IP address which is used on Public networks for communicating to millions of machines. The communication in a subnet is managed by a firewall configured and the network topology, meanwhile, machines on a subnet can fully communicate with each other while they may be restricted on other subnets on the same network.

* **Topology**

The Topology in a network map shows how subnets are controlled and connected to other subnets and can directly communicate among themselves with the subnets on the other networks. **In Figure 2-2,** a network topology is shown where the Subnets 1, 2 and 3 are connected to each other and can freely communicate data on this subnet while Subnet 1 has been configured to communicate with external subnets and networks. So, an attacker may get access to the et 1 to communicate and apply threats /exploits on the other Subnets (2 and 3) and a whole setup of machines in this network or his target point machine at this network. The subnets may be viewed as vertices with connections as edges while understanding the concept of this network topology.

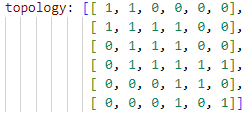


Figure 2-5. Example of topology definitions

Figure: 2-5 shows the example of topology definitions for the scenarios given in Figure 2-2. As we can see in the example, topology is given by the adjacent matrix which has the values of 0 for not connected links and 1 for connected links.

* **Firewall**

An important and key security component of a Network is Firewall. Firewalls are the components that exist in connections between the subnets as well as connectivity of machines in the network or external networks in an environment. The firewalls define which services are to be accessed or controlled while machines are installed in a network. Firewalls control the number of services accessible on a certain subnetwork. The firewalls allow selected services to be available on a subnet or accessible to another subnet from one machine to another, while blocking a certain subnet. The access of traffic to control in a subnet or associated subnets as blocked or permitted is defined in a Firewall set of rules.

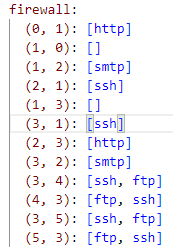


Figure 2-6: Example of firewall configuration in the network

In Figure 2-6, the configuration of firewalls for the network given in Figure 2-2 is given. The first number of each pair is the source subnetwork and the second is the destination subnetwork.

The accessibility is defined on ports however the rules of firewall on the subnet have been defined on the services rather than the network ports. The Firewalls are important because only selected ports can communicate the data or access a certain system.

* Scenarios

The environments described above are implemented using scenario files. Each Scenario file has the format of yaml and can conFigure all parts of networks. The scenario files which are used for testing are given in Appendix.

The Following Figures show the structures of networks used for testing.

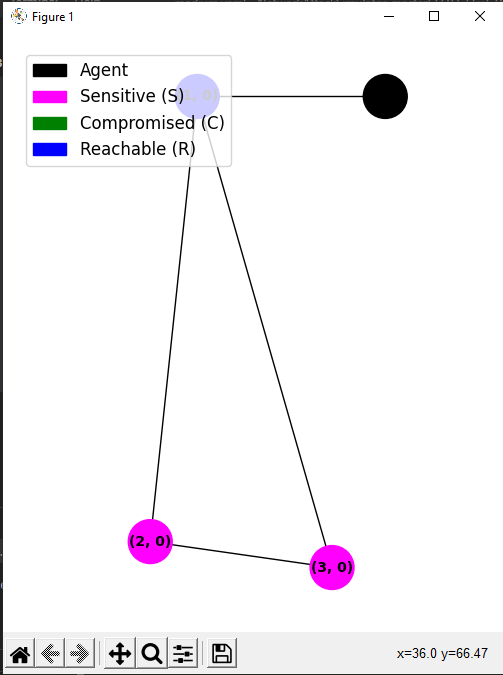


Figure 2-7. Initial graph of tiny Scenario

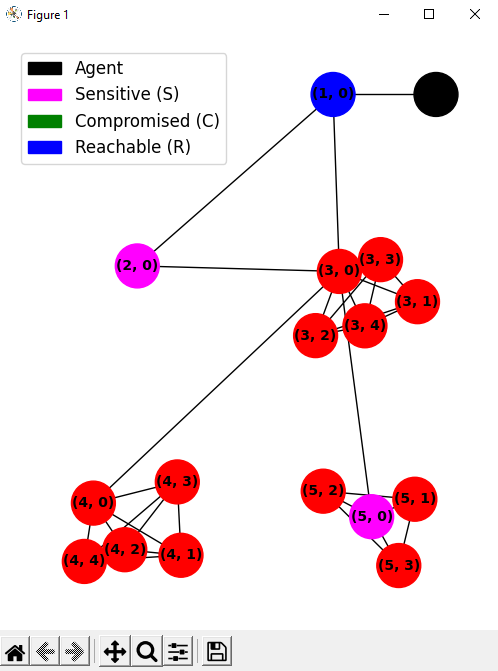


Figure 2-8. Initial graph of medium Scenario

The tiny scenario contains three subnetworks and three hosts.

The medium scenario contains 5 subnetworks and 16 hosts.

The numbers of sensitive hosts are 2 for both scenarios.

The optimal path for both scenarios also can be found in the scenario files given in the Appendix B. Connection information is also given in the scenario files in detail. The goal of a network attack is to compromise sensitive hosts.

1. ***Environments for DQN***

The Reinforcement Learning approach learns the best-required policies by interacting with the environment. It is performed by getting certain values randomly, then learning the data required to use for a given state, like Q(s, a), by selecting the task based on the current state of policy, implementing the action to the environment and updating the values Q(s, a) in state-action relying on the received response as in Figure 2-9. To learn action state values, Q-learning is a policy for temporal differences and it is the reason that Reinforcement Learning Implementations have a Q-learning mechanism. It is done by putting the values in the equation (2.1). Here, α is a step size that manages the amount of moving from the current estimate to the new estimate and is a discounting factor that determines how it is shifted from the current calculated value to the upcoming results. The calculated future rewards will be more accurate and it will specifically determine the optimal values. The quantity of visits to the state-action pair is determined by how Q-learning is displayed to meet the optimal action values.

(2.1)

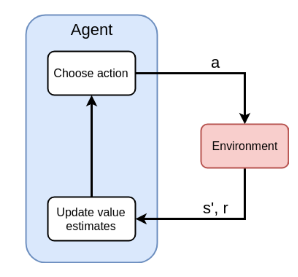


Figure 2-9. Training cycle of Reinforcement learning

RL algorithms are categorized into two main parts according to the determination method of value function Q (s, The first is a tabular method and the second is a functional approximation method. Tabular-based RL uses a table of state-action to determine the value function whereas the functional method uses the function. Figure 2-10 shows the determination method of the value function in the tabular method. As we can see from the Figure, the tabular method uses q-matrix to determine the value function from state and action. The Tabular method has the advantages of easily implementing and high performance to find optimal solutions for relatively small problems. However, it has the disadvantage of bad performance in the relatively large problem since it has to save all state-action pairs in the table. Functional approximation methods, on the other hand, have good performance for the relatively large problem and can generate inexperienced states too, although it is more difficult to implement than the tabular method and the performance is mainly affected by the selected function. DQN method is a main type of function approximation method and uses a deep neural network to approximate function. Figure 2-11 shows the determination method of the value function in the DQN method. As we can see from the Figure, the DQN method uses a neural network to determine the value function from the current state.

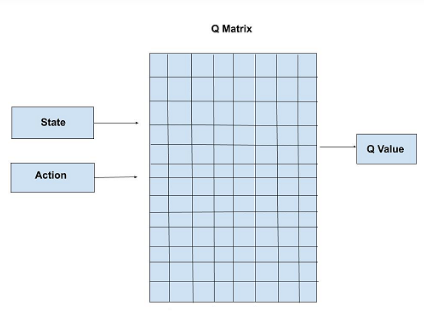


Figure 2-10. Determination of Q-value in tabular method

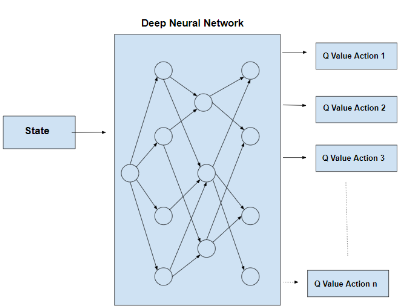


Figure 2-11. Determination of Q-value in DQN

For the high performance of neural network approximating functions, DQN gets high performance to large problems and has been accepting to find solutions to complicated problems.

* NAS: States

State (𝓢) is the set that contains the state of each machine on the network. The state of a machine includes its compromised or uncompromised status, its reachability or unreachability, and the status of its services (present, absent, or unknown). A machine is considered compromised if it has been successfully exploited, and it is considered reachable if it is directly connected to an external network or to a subnet that contains a compromised machine. The state space encompasses all conceivable combinations of service and machine states, including compromised, reachable, and service knowledge. [35]

The state space expands at an exponential rate in proportion to the number of machines and services present on the network. Equation (2.2) illustrates the state space's size, represented by | 𝓢 |, with |E| signifying the number of exploitable services and |M| signifying the number of machines. The exponential base is 3, which indicates that the agent's knowledge of each exploitable service can be either present, absent, or unknown.

(2.2)

**Figure 2-12** shows an example network and its corresponding state. The state reveals that the attacker has successfully taken over the machine at (1, 1) and it is sharing the information with Subnet 2 & Subnet 3 to communicate between the machines, as indicated by the true value of "reachable" for the installed machines in defined subnets above.

Furthermore, the Setup done for the Machine at (3, 1) is elaborated due to the action of scanning while the values set up in its configuration for machine 2 at (2, 1) remains are not known. Notably, the state does not incorporate any firewall settings since it necessitates privileged access to determine. Therefore, the simulator assumes that the attacker cannot obtain this information and must learn it indirectly through exploit actions' success or failure. For every service available or accessible on the network, an exploit action for the machine is available and this action is determinable depending on the environment chosen by the user and which service is running.

NAS: Actions

In the NAS, the action space consists of a scan action and a deterministic or non-deterministic exploit for each service and machine on the network. The scan action is modelled after the Nmap complete scan and always returns information about service presence or absence. Each exploit action's success is based on factors such as service presence, firewall settings, reachability, and success probability, and each action has an associated cost that can represent metrics such as time, skill, cost, or noise. The reward function in the NAS is defined over a transition (s, a, s'), where s is the starting state, a is the action taken, and ansos' is the resulting state. The reward for any transition is the amount of any additionally included machine in the future state minus the cost of the action. If no machine is compromised, the reward is simply the cost of the action. The goal of the attacker is to compromise all machines with positive value on the network while minimizing the number or cost of actions used. This aligns with the goals of a real-world attacker looking to gain privileged access or retrieve sensitive information, which can be set in the NAS by assigning certain machines specific values.

(2.3)

If there are any machines comparison, its value is returned in the value ((s’, s) in the above formulation from s in the s’ and when none of the machines was comprised, the value is zero and the cost of the action a is shown in .

Table 2-1. Network information for Tiny and Medium Scenarios

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | Type | Subnets | OS | Services | PrivEscs |
| tiny | Static | 4 | 3 | 1 | 1 |
| Medium | static | 6 | 16 | 2 | 5 |

Table 2-2. Information of RL Environment for Tiny and Medium Scenarios

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Actions | Observations | States | Step Limit |
| Tiny | 1 | 18 | 56 | 1000 |
| Medium | 3 | 192 | 493 | 2000 |

1. ***Experiments for DQN using NAS and Analysis***

* Deep Q learning algorithms

The DQN algorithm which is used for penetration testing is shown in the Figure. Line 1-3 initializes the parameters for DQL. Line 4 and line 6 show the iteration for episodes and steps. Each episode has several steps of action.

For each step, the algorithm first selects the action with the epsilon greedy policy and executes it. (Line 7-12)

Then it saves the transition into replay memory D and updates the weights of the neural network using gradient descent methods. (Line 13-17)

It terminates if the next state is terminal. (Line 18-20)

If the next state is not a terminal state, it updates the current state and goes to the next step. (Line 21)

|  |
| --- |
| Algorithm 3. Deep Q-learning with experience replay |
| 1. Initialize replay memory 2. Initialize action-value function Q, with random weights 3. Initialize target action-value function , with weights 4. For episode=1 to V: 5. =initial state 6. For step=1 to T: 7. If random(0,1)<epsilon: 8. 🡨random action 9. Else: 10. Select 🡨argma 11. End if 12. Execute action and observe reward and state 13. Store transition () in replay memory 14. Sample random minibatch of transitions () from replay memory if is terminal 15. Set 16. Perform a gradient descent step on with respect to weigh For 17. EveForrestt 18. If is a terminal state: 19. End episode 20. End if 21. +1 22. End for 23. End for |

* Steps for Testing.

Tests are divided into two steps.

Firstly we train the DQN agent and check if the training is converged and the goal is reached.

Secondly, we test the trained DQN agent using the same environment.

* Parameters for Testing

Table 2-3. Parameters used in training of DQN agents

|  |  |
| --- | --- |
| No | Parameter name |
| 1 | Scenario file name |
| 2 | Observe format |
| 3 | Size of hidden layer |
| 4 | Training steps |
| 5 | Target update frequency |

Table 2-3 shows the main parameters used in training the DQN agent.

For a tiny scenario, since there are 56 observations and 18 actions, there exist 56 inputs and 18 outputs for the neural network. There is no exact method to select the optimal number of hidden neurons. Any continuous Function can be approximated or predicted with a 1 hidden layer network, for a specific range; as stated in The Universal Approximation Theorem.

There exists several experienced formulas to determine the size of the hidden layer. The common principle to set the hidden layer size is that it should be between the input layer size and the output layer size. We set the number of hidden layers as 1 and the number of cells as 32.

Training steps are set as 10000 and the target update frequency is 1000.

Once training ends, we test the DQN agent. The Initial states of networks are given in the following Figure.

* Rewards

We used the rewards given in the NAS environment.

* Experiments

We perform experiments for tiny and medium scenario files using the DQN agent in NAS. Tiny and medium scenario files are given in Appendix B and the graphs for the initial states of both scenarios are given in Figure 2-7 and Figure 2-8.

In Figure 2-12, we can see the initial state of the tiny scenario.

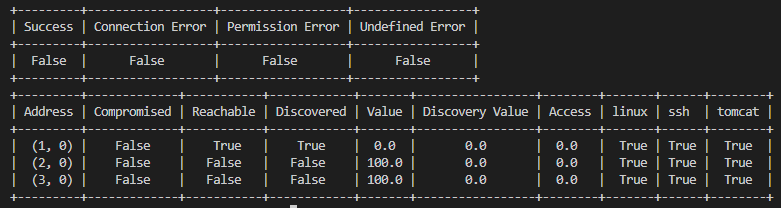
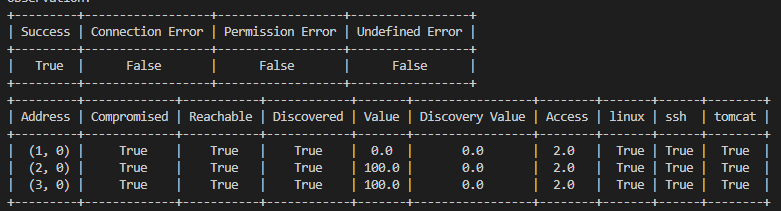


Figure 2-12. The Initial state of the Tiny scenario

The Final states of networks are given as follows.



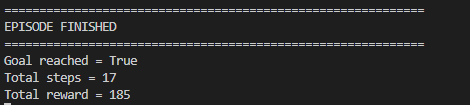


Figure 2-13. Final states DQN agent given in NAS for tiny scenario

Figure 2-14 shows the final graph of the tiny network.

As we can see from Figures 2-13 and 2-14, the DQN agent can find the attack path correctly and reach a goal. All sensitive nodes are compromised.

As we can see from Figure 2-13, it takes 17 steps to reach a goal.

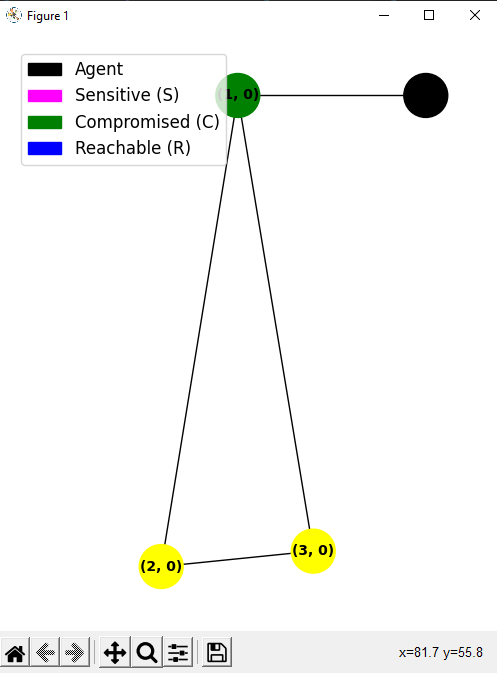


Figure 2-14. Final graph of the tiny scenario for the DQN agent given in NAS

Next, we test DQN agent for the medium scenario.

In this paper, we should several experiments for medium scenarios for comparison reasons, so we set the number of experiments and this is Experiment 1.

The structure and initial states of networks in this scenario are shown in Figure 2-15.

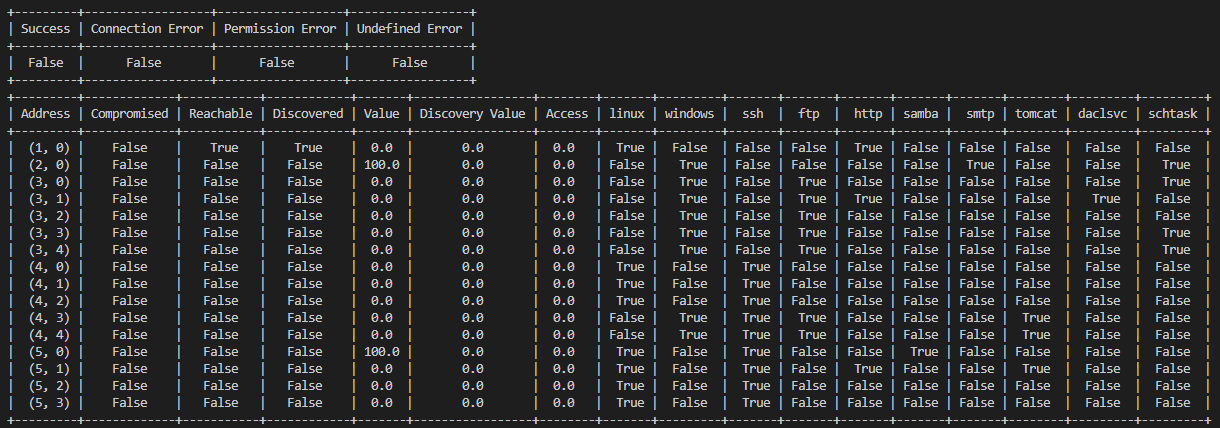
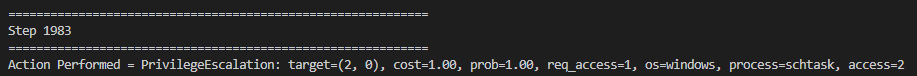


Fig 2-15. Initial states of medium scenario

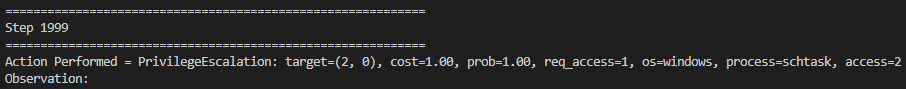
Since the input layer size is 493 and the output layer size is 192, we set the number of hidden layers as 1 and the number of cells as 256.

Training steps are set as 1000000.

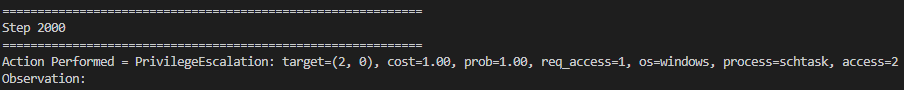
After training ends up, we test the trained model in the same way.



(a). Action in 1983rd step.

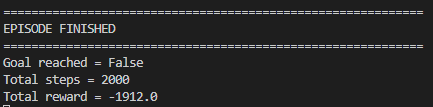


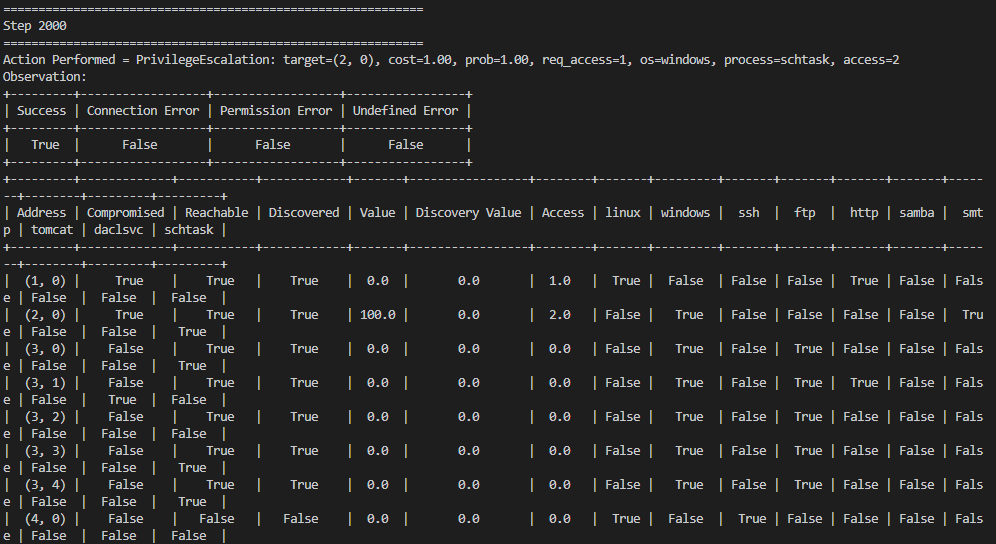
(b). Action in 1999th step.



(c). Action in 2000th step.

Figure 2-16, actions for the last steps





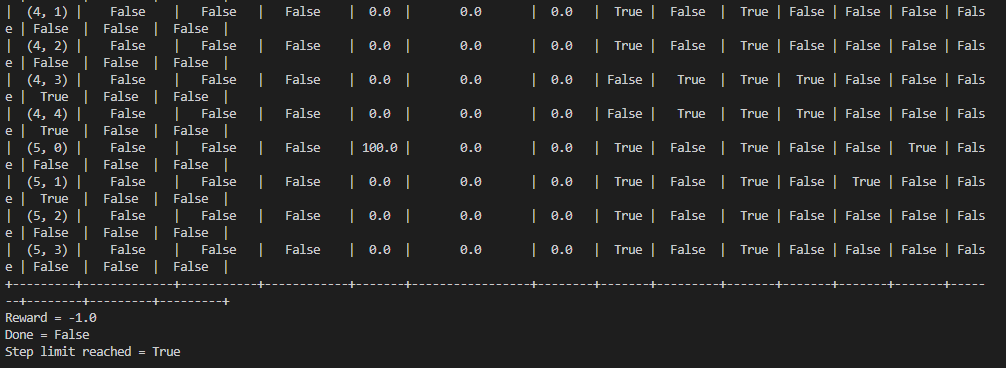


Figure 2-17. The Final state of network in Experiment 1.

Figure 2-16 ~ Figure 2-18 show the results of experiment 1. As we can see from Figure 2-17 and Figure 2-18, The test didn’t reach goals.

From Figure 2-16, we can see that agent selected the escalation for host {3,0} for the last steps. It means that the agent is put in an infinite loop. We performed several tests on this agent with the same environment, however tests showed similar results and the agent didn’t reach the goals.

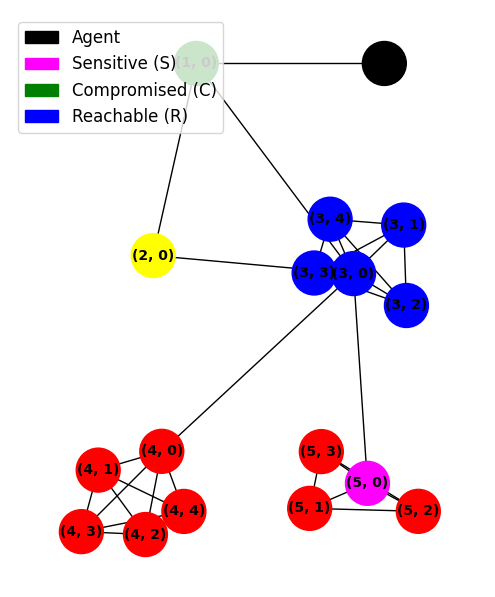


Figure 2-18. Final graph of network in Experiment 1.

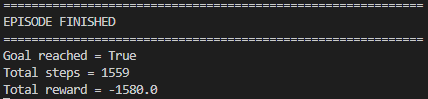
As we showed in the algorithm, we use the epsilon greedy selection method to select action, however since epsilon is not large, the agent gets difficult to avoid infinite loop.

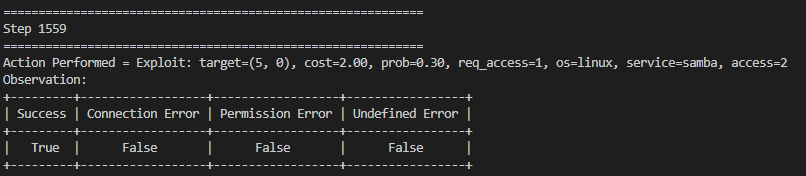
To prevent this, we can set the epsilon as a larger value to select arbitrary action instead of action with maximum q-value. Since we select the arbitrary action with epsilon probability, if we increase it, the probability to select arbitrary action will increase.

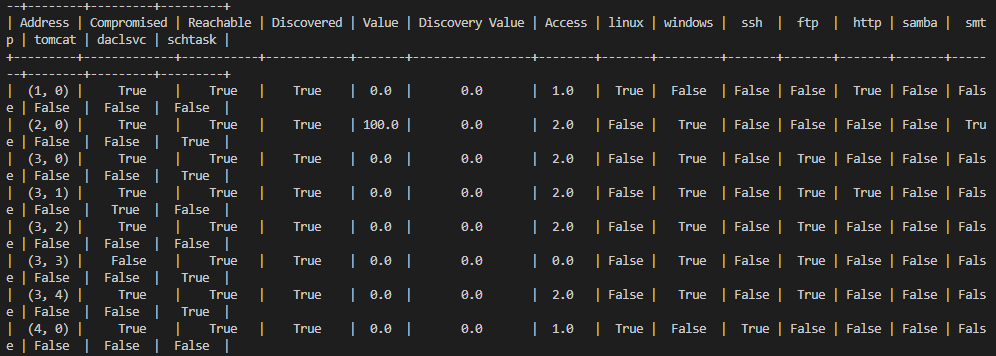
Another issue will be generated if the value of epsilon is set too high as it will downgrade the performance of the agent. The probability of selecting a random action will be increased and the agent’s performance to avoid an infinite loop will be upgraded.

We perform Experiment 2 to avoid an infinite loop by selecting a random agent if the same action is selected repeatedly. We select the epsilon value as 1 if the same action is selected more than three times repeatedly.

Results for this case are shown in Figures 2-19 and 2-20.







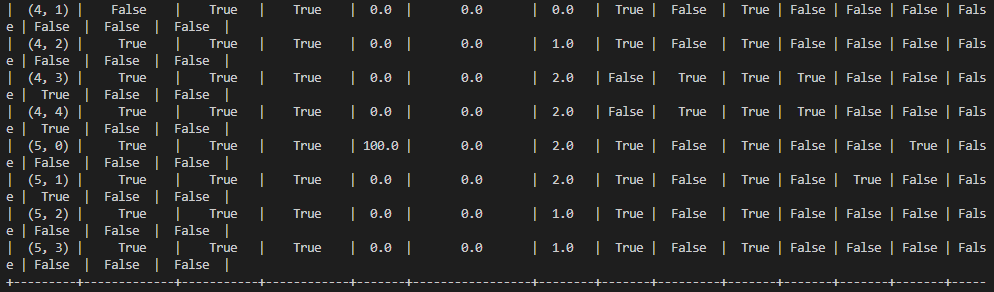


Figure 2-19. The final State of the network in Experiment 2.

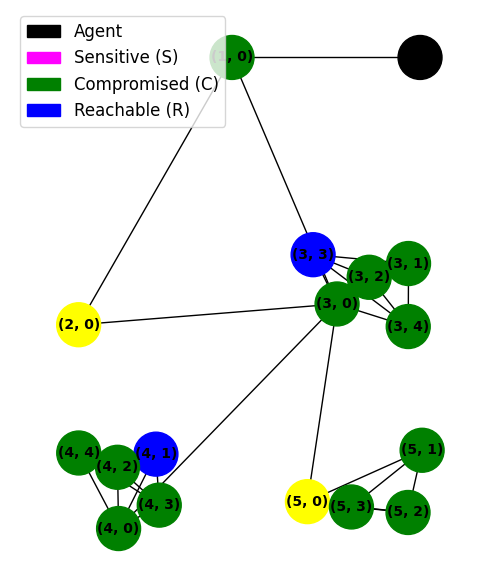


Figure 2-20. Final graph of the network in Experiment 2.

As we can see from the Figures, the agent avoided the infinite loop and reach a goal. However, we can see that steps to reach a goal gets 1559 steps.

In fact for this scenario, the length and reward of the ideal optimal path are 10 and 192. The Optimal path is given in the scenario file in Appendix B. We performed experiments for this agent several times and get similar results.

Table 2-4. Total steps and Reward of Experiment 2 and [1]

|  |  |  |
| --- | --- | --- |
|  | Total Steps | Total Reward |
| Experiment 2. | 1371.45+/- 420.41 | -1875.29+/- 660.62 |
| Preceding[1] | 1401.34+/- 317.23 | -1902.32+/-603.47 |

As we can see from Table 2-4, the result is similar as given in [1]. Although a goal is reached, the length of the path is too long than the optimal path and the reward is also too low.

Tests showed that the DQN agent in the NAS has good performance for tiny networks but has some problems to be solved for big networks.

In the next sub-section, we are going to try to analyse the reasons for these problems and find the solution to solve them.

* 1. **Analysis**

In the previous sub section, we showed the tests for the DQN agent in the NAS environment and the DQN agent has bad performance for big networks.

As we described in the above sections, the number of states and actions is increasing rapidly with the number of hosts. Net experts who have full knowledge of network attacks can find the attack path for the scenario used in the preceding section since they can find current available attacking methods based on observation.

This means that if we use the information for network attacking methods, we can upgrade the performance of the DQN agent largely.

This can be implemented by changing the reward function of the NAS environment. In previous sections, we described for NAS environment.

To improve performance, it is needed to reflect the information of net attacking methods and this can be implemented by updating the environment for rewards.

The objective of upgrading is to reduce the length of the attacking path. We implemented an update for the algorithm in two steps. The First step is to study for NAS environment and find its drawbacks. For this step, study for reward function seems most important.

The Second step is to make updating the project based on the result of the first step.

Updating for the NAS environment is described in the subsection.

1. Updating Performance Auto penetration testing based on DQN agent
   1. **Study on NAS environment**

We described the structure of the NAS environment in the preceding sections. In this subsection, we are going to study for Reward function of NAS critically.

The reward of NAS is separated into two parts. The first part is the value which represents that node was compromised or not compromised the and second part is the cost which represents the time to do such action.

As we can see easily, the second part is to reduce the length of the path and find the optimal attacking path. This worked well for tiny network however not worked well for a big network as we described in the previous section. The first part is used to make the DQN agent know the information if the action compromises the host. It makes the DQN agent select the action to have a high probability to compromise the hosts. It is set as 1 if the host is compromised and 0 if not. This term is added to make DQN agent to be converged quickly. However as we described in the previous sections, the DQN agent didn’t work correctly as expected. It is because the agent didn’t use the information of network attack methods.

Table 2-5 shows the evaluation of value which is the first term of the reward function given in equation (2.3) in the current reward function. Table 2-6 shows the evaluation of cost which is the second term of reward function in the current reward function.

Table 2-5. Evaluation of Value for actions in the original reward function

|  |  |
| --- | --- |
| Case | value |
| Action compromise agent | 1 |
| The action didn’t compromise | 0 |

Table 2-6. Evaluation of Cost for actions in the original reward function

|  |  |
| --- | --- |
| Case | cost |
| Exploit(e\_ssh) | 3 |
| Exploit(e\_ftp) | 1 |
| Exploit(http) | 2 |
| Exploit(e\_samba) | 2 |
| Exploit(e\_smtp) | 3 |
| Privilege escalation | 1 |
| Service scan | 1 |
| OS scan | 1 |
| Subnet scan | 1 |
| Process scan | 1 |

From the tables, we can see that the reward function can only represent whether the information host is compromised or not but detailed information such as which action is a success. So DQN agent can’t decide the correct action for a given environment.

For this reason, training for DQN agents is not performed correctly and the second part of the reward is not worked correctly. After all, for most cases, rewards are decided by second part mainly and DQN selects the action which has the lowest cost for all cases.

This is the reason that DQN agents worked badly for big networks. From analysis for the DQN agent, we can know the fact that values for reward should be set considering attacking methods second term for cost can’t work for a big network.

* 1. **Updating of DQN agent and NAS environment**
* Study on attack methods

A Network attack has 6 steps Exploits, privilege escalation, subnet scan, os scan, process scan, and service scan on NAS.

The First step of attacking is to exploit several protocols such as ssh, FTP, HTTP samba and so on.

Once the host is exploited, we scan for the subnet in which exploited hosts are contained. Privilege escalation is used to attack sensitive hosts and should be performed once the exploit to the host is successful. Os scan, process scan and service scan are used to scan protocols and os.

Network experts find reachable hosts and exploit them. Since network experts are based on information for knowledge, they can’t jump any steps for attacking. For example, network experts will not try privilege escalation for not exploited hosts.

We should reflect this information to the NAS environment to make DQN agents this knowledge.

From the study of the current NAS environment and network attacking methods, we propose a new reward function for NAS and verify performance through experiments.

From the network steps, we set the rewards as follows.

Table 2-7. Evaluation of Value for actions in the new reward function

|  |  |
| --- | --- |
| Case | Value |
| Action is failed | 0 |
| Service Scan success | 2 |
| Os scan success | 1 |
| Exploit Success | 4 |
| Privilege Success for general host | 3 |
| Privilege Success for sensitive host | 100 |

As we can see from the table, the reward value is set to reflect the network step.

The Exploit is the most important step and also the first step. Since a successful exploit gets a greater reward value than others, the DQN agent avoids being put in an infinite loop.

Privilege Success for a sensitive host gets the greatest reward which is much larger than other rewards.

It is since attacks on sensitive hosts are our goal.

It is expected that once we use these rewards, training can get better performance than the preceding one.

The problem here is that we can’t consider the cost of attacking correctly so it is difficult to reflect the optimization of the attacking path.

We are going to solve this problem by applying cost value as in the [1].

However, we don’t use the cost value as in [1]. Since we set the value based on network attacking methods, there are no means to set the cost differently for the attacking method.

We are going to set the cost as a bigger value than the reward value for successful attacking steps except for escalation to sensitive hosts.

If then, rewards for all cases except success privilege escalation to sensitive hosts will be negative.

Attacking with longer lengths will have smaller rewards. So we can see that training for the DQN agents will be performed in the direction to reduce the length of the attacking path. Though this method can’t reflect the cost of attacking, it will help to reduce the length of the attacking path. As we previously described, the cost of current rewards can’t help the DQN agent to find the optimal path for big networks. It can only work correctly for a tiny network. For big networks, there are too many actions, DQN agents should consider the network attacking methods to reduce the possible actions and also find the smaller path possible.

We are going to check the performance of the proposed method using experiments for the same scenario.

To see the effects of proposed reward function, we first use only the first term of reward in the first experiment. Next, we apply the proposed reward to the NAS environment and get the results.

We are going to compare the performance of experiments.

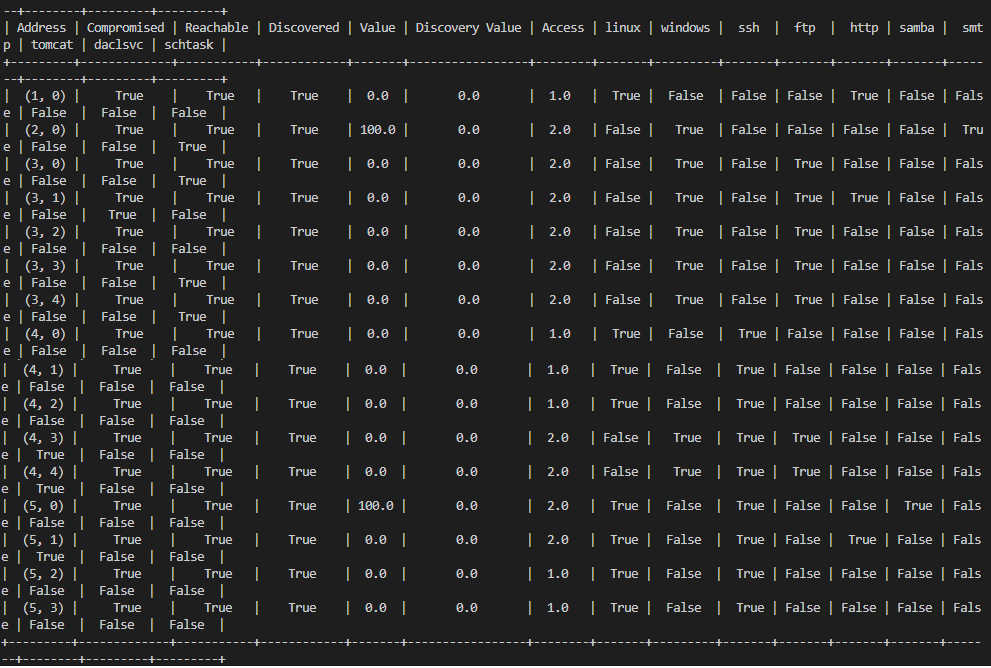
# Experiments / Testing / Measurements / Data

Firstly we perform the test with only the first term of reward and it is Experiment 3. We expect that the training will be a success in making DQN agents reach a goal. We change the value of the reward function in the NAS environment as in Table 2-7 and set the costs for all actions as 0. We perform experiments with medium scenario file. In experiments 1 and 2, we recognised that if the values for actions are not set correctly, although we set the costs for actions properly, we could not get the expected results. For this reason, we are going to experiment based on setting costs for all actions as 0.

In the next experiment, we adjust the costs as proper values and compare the performance for both experiments to see the effect of costs on the performance of the DQN agent.

The training steps are set as same as in the previous experiments and other parameters are also the same as in the previous experiments. We use the proposed reward in the training step only and use the original reward in the test step to compare the performance with the original.

Figures 2-21 and 2-22 show the result of Experiment 3.



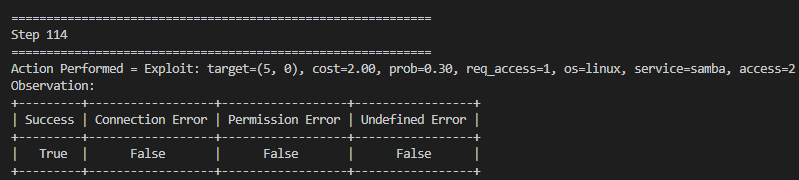




Figure 2-21. Final state of network in Experiment 3.

As we can see from Figure 2-21, it takes 114 steps and the total reward is 76. All nodes are compromised in this experiment as we can see in Figure 2-22. It shows that the proposed algorithm gets perfectly efficient to make the DQN agent find the attack path.

The problem here is that our object is to compromise sensitive hosts as soon as possible. Compromising all hosts to get sensitive information will take much cost.

Compared with Experiment 2, it takes much smaller steps and compromised all hosts in the network. It shows that considering attacking methods in the reward function makes the DQN agent compromise hosts. It means that the reward values we proposed work correctly as we expected. On the other hand, it shows that the DQN agent in Experiment 3 couldn’t find an optimal solution with minimum path length.

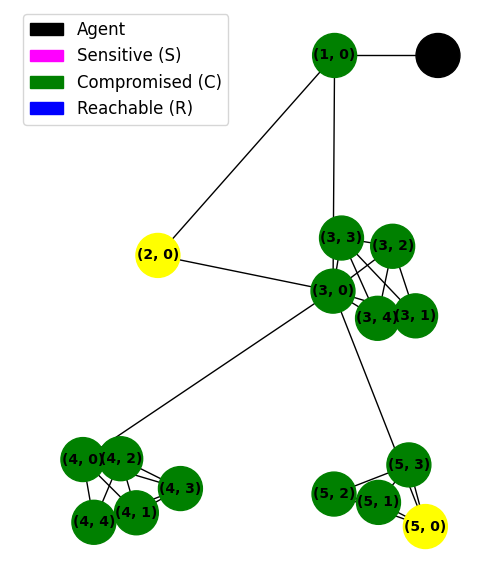


Figure 2-22. Final graph of network in Experiment 3.

To analyse the performance of the proposed reward, we performed 10 experiments and showed the results in the following table.

Table 2-8. Results of 10 experiments in Experiment 3

|  |  |  |  |
| --- | --- | --- | --- |
| No | Total Steps | Total  Reward | Goal Reached |
| 1 | 114 | 76 | Yes |
| 2 | 241 | -53 | Yes |
| 3 | 251 | -61 | Yes |
| 4 | 135 | 57 | Yes |
| 5 | 273 | -83 | Yes |
| 6 | 37 | 155 | Yes |
| 7 | 184 | 12 | Yes |
| 8 | 81 | 109 | Yes |
| 9 | 58 | 128 | Yes |
| 10 | 61 | 127 | Yes |

As we can see from the table, the goal is always reached and the total steps are in range (of 37,273) for 10 times experiments.

Total Rewards are in the range (-83 to 127).

As we described in the previous experiments for original rewards, total steps are reduced 10 times at least and total rewards are increased largely.

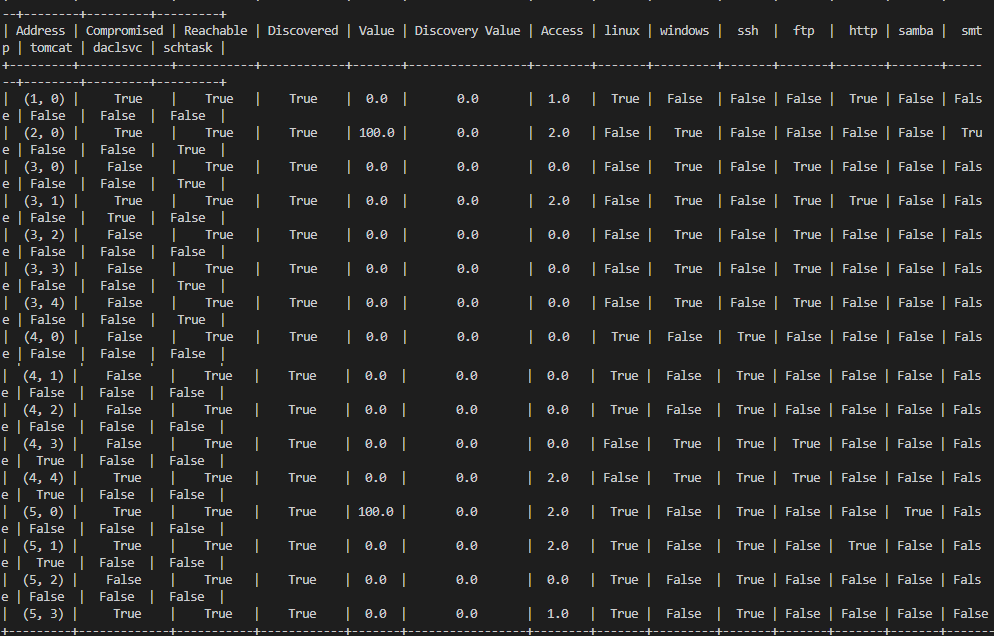
Now, we can go into the next experiment, Experiment 4.

In the next experiment, we consider the constant cost in the reward function and rewards for all actions except for success privilege escalation to sensitive hosts get negative rewards to find a small attacking path.

An Increment in attacking path length will produce decrement in total rewards and the DQN agent will be trained to find the small attacking path.

We experiment in the same ways as in Experiment 3, however, we consider the second part of the reward. Here, we set it as a constant value of 5. If then all actions without successful privilege escalation to the sensitive hosts would be negative value and training will be performed in the direction that number of actions taken is reduced since an increment in action numbers taken will decrease the rewards. It is similar to in [1] however the rewards for successful actions are selected as different values considering the attacking procedures and are expected to upgrade the performance of the DQN agent.

In this experiment, we expect that the DQN agent will reach a goal with smaller actions. It will be represented as the smaller number of compromised hosts in the final graph of the network.



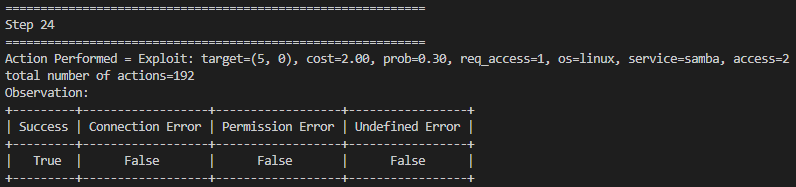




Figure 2-23. Final state of network in Experiment 4.

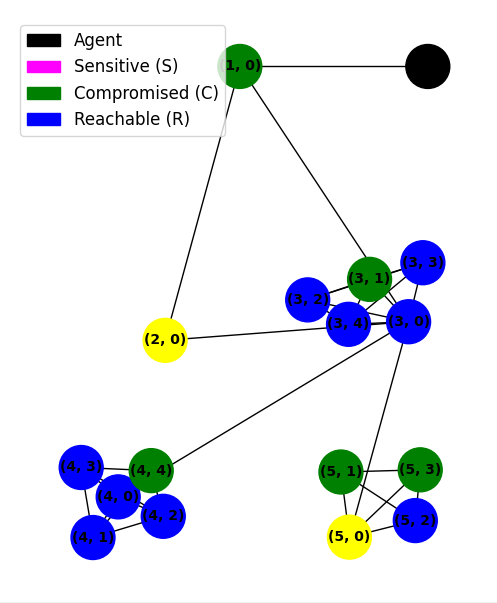


Figure 2-24. Final state of network in Experiment 4

Figure 2-23 and 2-24 show the result of Experiment 4.

As we can see in Figure 2-23, the total steps are reduced to 24 and the total reward is increased to 172. Compared to the ideal optimal reward 192, it is a very high reward value. The Total number of steps is only 24. As we can see from Figure 2-24, there are smaller numbers of compromised hosts than in preceding experiments as we expected.

Table 2-9. Results of 10 experiments in Experiment 4

|  |  |  |  |
| --- | --- | --- | --- |
| No | Total Steps | Total  Reward | Goal Reached |
| 1 | 24 | 172 | Yes |
| 2 | 35 | 159 | Yes |
| 3 | 61 | 131 | Yes |
| 4 | 43 | 149 | Yes |
| 5 | 18 | 178 | Yes |
| 6 | 144 | 46 | Yes |
| 7 | 130 | 58 | Yes |
| 8 | 22 | 170 | Yes |
| 9 | 100 | 90 | Yes |
| 10 | 57 | 141 | Yes |

For validation, we perform 10 experiments with the same condition as same as in the first experiment.

Test results are shown in table 2-9.

# Analysis and Discussion of Results

In this subsection, we analyse the performance of proposed algorithms with experiment results.

We calculate the mean, and deviation for rewards and steps for experiments 2-4 and compare them. Experiment 1 doesn’t reach a goal, so we don’t show its result in the table. Table 2-10 shows the performance of several methods. The results are calculated from the results of 10 experiments.

Table 2-10. Comparison of total steps and rewards

|  |  |  |
| --- | --- | --- |
| Method | Total steps | Total rewards |
| Preceding [1] | 1371.45+/- 420.41 | -1875.29+/- 660.62 |
| Experiment 2 | 1401.34+/- 317.23 | -1902.32+/-603.47 |
| Experiment 3 | 143.500+-27.84 | 46.700+-27.72 |
| Experiment 4 | 63.400+-14.47 | 129.400+-15.21 |

In Experiment 1, we perform experiments using the same reward function given in the NAS environment and recognise that it can’t find the attacking path for the medium scenario since it can’t avoid an infinite loop.

In Experiment 2, we used the same reward function however we changed only the epsilon value to avoid the infinite loop.

It reached a goal and has similar results for total steps and rewards. We could validate that the results given in [1] are based on this method.

Experiment 3 uses the upgraded rewards given in Table 2-7, considering attacking procedures and shows more excellent performance than the preceding ones. In Experiment 3, all the rewards are positive so the path with a longer path could have larger rewards. This takes negative effects on the performance of the DQN agent to find optimal solutions which reach a goal. Our goal is to compromise all sensitive hosts with minimum attacking path.

To solve this problem, we performed Experiment 4 where all rewards except the case of success privilege escalation to sensitive hosts are negative. From equation (2.3), we can see that rewards in NAS have two terms and in Experiment 3, we used only the first term. In Experiment 4, we set the second term of rewards for all actions as 5.

As we can see from the table, the result of Experiment 4 reaches a goal with smaller steps than Experiment 3 and total rewards are increased. Results of experiments show that prosed algorithm can upgrade the performance of DQN agents largely.

DQN agent has full ability to find the attacking path with small total steps and high total rewards.

Experiments show that the reward function affects the performance of the DQN agent mainly.

# Conclusions

In this paper, we review the research for auto penetration testing based on DQN agents and find the drawbacks. The aim of this study was to know about the application of penetrating testing in a simulated environment and to get optimal cases that may incur if a network or machine available on this network is exploited or comprised by an attacker. It is demonstrated that it is possible to utilize RL in a simulated environment to determine the system strength and optimally exploit the network. Through this method, it will be possible to determine which parts of the network are including vulnerable points and patching is required for the security of the network. The major advantage of using RL as compared to other Artificial Intelligence Techniques is that RL doesn’t need any previous knowledge of exploits models and a defined standard algorithm to be applied and tested on different networks and network maps, either small setups as in this test simulator or big organization networks.

We proposed a new algorithm to update the performance of DQN agents reflecting network attacking methods.

We implemented a new algorithm on the NAS environment and are extendible to other auto penetration tools.

We concentrate on the upgrade of DQN agents to work well for big networks and find the short attacking path.

From experiments, we validated that the proposed DQN agent can work well instead of network experts for big networks. Since our research is concentrating on the reward function, we can extend this method to all Q-learning methods applied in network attacking simulation.

It is expected that the proposed method can be applied to all reinforcement learning-based auto-penetration testing methods. It also can be applied to large networks since it makes the DQN agent find the problem and report it.

# Future Work

In this research, we can see that in applying Reinforcement learning methods to auto penetration testing, the environment has a very important role.

We concentrate on updating of Reward function in NAS.

If we consider the network attacking methods to the reward function, we can upgrade the performances of DQN agents of other auto penetration tools.

We are going to study network attacking methods to other penetration testing tools which use RL methods.

Appendix A: Literature Review

This project has extensive roots and aligns with the applied research in streamlining and enhancing offensive cybersecurity auditing techniques, with a notable focus on Vulnerability Assessment (VA) and Penetration Testing(PT) [5,14]. In this context, the authors provide a summary of previous research efforts, highlighting the adopted approaches and significant contributions.

Initially, researchers were focused on the planning phase of these processes, with some works implemented in industrial PT systems and frameworks, while others remained in the realm of research ideas [14, 16]. As the domain of PT automation and enhancement lies at the intersection of cyber security and AI research fields, multiple avenues of research have been pursued through different methodologies of automated planning (a sub-area of AI) [18,19].

Early research on this topic was about modelling penetration and how penetrating testing is used in the decision-making approach [20]. Although most of the works were initially focused on vulnerability assessment, they paved the way for significant contributions to the PT domain as well.

To provide a comprehensive understanding of the research in this field, the paper presents a literature review section summarizing previous research efforts, with a focus on the adopted approaches and contributions. To get an optimal result, the authors today present the actual results of the search in this field and then further divide it for research in approach, methodology and by type. [18].

1. **A detail of previous works done on Penetrating Testing automation**

When the object of a system is to develop an efficient PT system for the organization, the most famous approach for PT tasks is automation.

If it is don’t in an appropriate way, we may fail to achieve the goals as each phase of automated testing for main tasks and their sub-tasks or activity testing is highly challenging, e.g. the utilization of automated systems and tools that may be performing their tasks, their actions and the results without any pre-processing or optimization may be a huge failure to get the optimal results. The automated systems may not be able to produce the required results in small or large environments if there is no permanent control of a PT human expert reviewing the process. It is because automated PT requires a large number of complete network structures [20-23]. In Penetrating testing environments, there are a number of issues that incur during the process in addition to the actual time duration of the tests. The issues which are highlighted to automation are a big number of false positive alerts notified on the assets such as firewalls and IDPS and network congestion or generated traffic. Therefore automated Penetrating Testing was only limited to networks of small size or medium size and those with customized scripts to perform only Penetrating testing. [25]

Initially, research focused on improving the PT system by enhancing the planning phase, which was approached through attack graphs or decision trees, reflecting the PT practice's nature as sequential decision-making. However, most of the works focused on vulnerability assessment rather than PT, as the approach was static and limited to the planning phase. One notable contribution was the modelling of VA as atomic components, pre-conditions, and post-conditions to narrow down the targeted vulnerability. Later works attempted to automate the planning phase of PT tasks, but this approach did not address the issue of improving performance and only focused on the planning phase.

Notwithstanding, one remarkable work introduced optimization by modelling PT as planning domain definition language, which calculated the Penetrating Testing Post-attacking and attacking phases, allowing for integration with some PT systems. This solution generated different types of attack plans for single and multi-paths PT scenarios and was executed in conjunction with information-gathering tools, which transformed the acquired information into input for a planning problem to be solved separately. However, the proposed approach was limited to small and medium-sized networks. [26]

Some research considered AI to enhance PT practice, but most of the proposed models were not successful to cope with continuous uncertainty in Penetrating Testing practices and the unavailability of reliable and accurate knowledge about the systems assessed. A possible way was to use the ML algorithms in the VA System and skilled Penetrating testing called Core-Impact, which modelled the PT planning phase as a partially observable Markov decision process resolved with a solver from external POMDP to calculate the best testing plan in the form of attack vectors. However, this model was insufficient to include the pending tasks and testing phases, such as pivoting, testing and vulnerability assessment that should be non-standard, sequential and highly interactive when matched with information collection starting from the planning phases. [27]

1. **Major Drawbacks and Limitations of Current Penetrating Testing Practice**

We will present the current Penetrating Testing Practices, tools, systems and frameworks, limitations and major drawbacks of these practices along with the domain of penetrating testing and automating PT Practices. On Large networks, the penetrating testing becomes slow because it includes repetitive and routine tasks. Although much of the penetrating testing routine can be automated, the repetitive tasks are crucial and can’t be skipped resulting in slowing down on large networks [28]. While they were in theory and in the documents, many of the proposed solutions were looking effective and to help solve the problem, the practical demonstration of penetrating testing proved that added improvements were not good enough to solve the core issues along with available resources and infrastructure.

On the other hand, some solutions were not adequate in the context of penetrating tests. It is evident today that human computing capabilities are limited and time-consuming as compared to machines when it comes to today’s complex tasks and large algorithms. It takes days or weeks for an average Penetrating tester while work on a medium-sized Local Area Network and the core focus is that testing on a specified network should be comprehensive and cover all available services and machines on that network. Along allocate time and effort, a large amount of system downtime will be reported while performing the tasks. In terms of accuracy and quality of the results, the above two points add to the poor performance and it may be a fact that human beings change their opinion, get bored and commits mistakes. [29]

In response to the above issues, **Penetrating Testing Automation** was presented as a perfect solution. Therefore, semi-automated or fully automated solutions were designed with their main goal to reduce the frequencies of testing, testing coverage increase and save time and reduce human labour. [30] In terms of a defined approach, the proposed solution was very effective and adaptive to rely on automated planning to generate an automated attack plan. This attack plan was named an attack graph. Meanwhile, the newly added system solutions worked to perform complex tasks and shorten the overall process along with adding new cases of exploits.

The Existing Penetrating Testing tools, frameworks and systems could be easily automated and therefore the limits of these existing PT Systems were the major questions of the Cyber threats security community. These PT Systems can perform the majority of PT Testing tasks with little or no human involvement, especially during the discovery of Vulnerability and information gathering- the first two phases of Penetrating Testing. [31]

The companies which have a constant need for their security audit, the fully automated Penetrating Testing Systems which may perform basic and advanced testing without any human presence are more demanding to them. In spite of that, the researcher had been in continuous struggles to find Penetrating Testing Tools which may automatically perform their sophisticated procedures as humans can hardly master the processes due to the extent of their complexity. Therefore, developing a machine that would replace the PT experts is a major challenge and interest of the researchers also because PT practice has multi-phase nature with dependency on tasks and phases of the testing. [32]

The process of performing a PT (Penetration Testing) can be complex, and one major issue is the incomplete profile of the assessed system that is produced during gathering the information and reconnaissance phase of the PT. Experts often deal with this issue by repeating tasks, changing their approach, or making assumptions to continue tests. Additionally, modern attacks have adopted evasive techniques and complex paths, allowing them to bypass network and system defences. Skilled attackers exploit a chain of vulnerabilities, including hidden and composite vulnerabilities, in networks, making it difficult to secure the entire infrastructure. [33]

In the end, the penetrating Testing Data output is important, but it is often not utilized at the time of future tasks or retesting and it is put aside after the generation of the PT Report.

Our research aims to address this issue by focusing on using previous test results when retesting is required, as many of the system architectures and security configurations do not change remarkably in the short or long term.

1. **Motivation and Contribution**

Computer systems are no exception to the rule of High Complexities threatening the controls and security of the systems. It is a general understanding that increased complexity becomes a threat to the existing Ecosystem of Security and Controls and the same applies to computer systems. The Security professional avoided the trap of bolting policies and security layers; a classic approach of defending digital assets and networks from external cyber security threats and they focused their thoughts on offensive security.

The Security Professional was defined with the requirements of an Intelligent Penetrating Testing framework and System which should help with manual human testing work. There was much focus on the complexity of the testing systems and the high demand of PT to perform more or less automated tasks while taking it over from human interference and conducting data collection, assessment of vulnerabilities and risk factors, and exploits with more time to Security Professional to research and work on complex solutions of Post-Exploiting results. [34]

Based on our mentioned facts about Penetrating Testing practices for evaluating the systems

Security, Machine Learning (ML) is found to be the best solution to that problem. In the initial phases, many Artificial Intelligence (AI) techniques were introduced, but Machine Learning was the most sophisticated approach selected for the automated PT system and acts like a professional security Tester learning and gaining skills over time and implementing it in real-time testing scenarios.

This research will highlight the gap between current Penetrating Testing practices and focus on solving the following key issues:

* To reduce testing system cost due to Human labour costs and move towards systematic testing as an alternative to it.
* To allow adaptability and flexibility while dealing more affectivity with different types of known security threats.
* Significantly mitigate security risks, improve performance, and minimize downtime during testing to minimize the impact on the assessed network.
* To free Testing Experts from repetitive and boring testing tasks and assign them more challenging and purposeful assignments.
* To perform a wide variety of security exploits and make use of evasive and complex paths of the network which may be hard to find if investigated manually by the human.

We may summarize that cyber attackers aim to get specific access through exploitations of the installed systems by going through a chain of attacks and finding any possible vulnerability.

The objective of IAPTS is to automatically detect, utilize, and manage network machines and networking devices while uncovering concealed and intricate security flaws within network infrastructure or segments. In essence, IAPTS should encompass all phases of penetration testing, going behind the vulnerability assessment offered so far.

The systems which prove to be secure while working in individual control also help to cover the testing and secure the locally secured systems but the addition of both may result open a loophole for the hacker to exploit through it. In the end, the Penetrating Testing expertise re-utilization, generalization and extraction is the concluding reward of IAPTS to the prevailing Penetrating Testing Practices.

The realm of computer science and artificial intelligence (AI) has witnessed a significant increase in research related to the creation and evaluation of intelligent decision-making systems. IAPTS is a system that perceives its environment and independently determines the most efficient approach to conduct PT tasks. Cybersecurity solutions powered by AI typically fall under two categories: expert-led and automated systems that use unsupervised machine learning. The former, such as AVs, FWs, IDPSs, and SIEMs, rely on input from security professionals and may be prone to errors.

1. **Reinforcement learning (RL)** techniques guide the existence of autonomous or semi-autonomous decision-making systems that reflect real-world contexts, particularly in the offensive security domain, such as vulnerability assessment and PT context. RL was selected for its capacity for autonomous learning, reward-based learning, and the richness of its environment, which can capture all major features of PT, including uncertainty and complexity. Reinforcement learning is a branch of AI that enables software agents to determine optimal behaviour within a given context to maximize performance. The agent adjusts its behaviour based on simple feedback (reward) from the environment, learning and adapting to improve its decision policy, if necessary. The best approach to maximize the overall reward is by some Reinforcement Learning Algorithms which can cover and optimize to global requirements.

The RL environment is explored, and the agent learns how to act based on the rewards received from its actions.

Human Intervention is significantly reduced or totally removed with the use of the Reinforcement Learning method. Furthermore, the time required to learn and customize is less as is the case with expert systems and machine learning correlatively. The suitability of Reinforcement Learning for increasing the automation of Penetrating Testing solutions has been greatly tested as said above and many new algorithms have been tested and added. These new algorithms are implemented along with new toolboxes and enhanced usability to answer complex problems of RL in restricted resource scenarios with many effective results.

* 1. **Towards Partially Observable Markov Decision-Making Process(POMDP)**

Partially Observable Markov Decision Process(POMDP) is another approach that involves modelling an attack on a computer system with an attacker’s incomplete knowledge in the simulation environment with configuration and learning to threats continue as the threats are applied to the system. [35] It is effective when applying Penetration Testing on a single host Mac3hine.

The concept of Penetration Testing (PT) involves a series of tasks that can be carried out manually or by an automated PT platform to achieve a predetermined or unknown goal, which is referred to as the target. The target could be a computer system, network, or information stored on a computer. In some cases, the attack may not have a specific target. The goal of this study is to create an efficient and optimized autonomous PT system that utilizes Reinforcement Learning (RL) and other techniques to enhance its performance, efficiency, testing coverage, and reliability. To model PT as an RL problem, the researchers employed the Partially Observable Markov Decision Process (POMDP). The POMDP model comprises a tuple M=<S, A, O, T, Ω, R,b1> with finite states, actions, and observations in S, A, and O, respectively. The agent interacts with the environment by executing an action and receiving an observation and a reward, without directly obtaining information about the state of the environment. The choice of using POMDP for PT modelling was previously explained in a publication, and some of the key points are summarized in the following sections. Since an attacker would have incomplete knowledge and is not sure about which of the hack scans may succeed while performing in on the system, the POMDP approach helps in defining the most relevant properties of real-world hacking.

The use of Partially Observable Markov decision processes (POMDPs) has proven to be a powerful tool in solving planning problems that involve hidden states and uncertainty in action effects. Recently, algorithms that employ an approximate value iteration technique with point-based update updates have been developed, which have been approved to measure correctly.

These algorithms depend on executing several highly quick approx. updates over a set of beliefs drawn from the reachable part of the belief simplex.

By calculating a bound on the estimated error that is proportional to the sample spacing of the set, one can tolerate more calculation error at the places that can only be reached after many time steps in discounted problems. This idea is based on a convergence result that is derived from the fact that the set samples enough of the search tree to some depth. However, fully expanding the search tree to depth t requires a number of points exponential in t.

This article introduces a novel convergence argument that combines elements from both approaches and applies to cases where the varying sample spaces are according to what is known as discounted reachability. By more accurately reflecting the behaviour of current algorithms, this new approach is expected to provide improved performance and efficiency.

**Partially Observable Markov Decision Process (POMDP) Solving Algorithm**

There is a complex sequence of decisions in Reinforcements Learning (RL) methods while solving real-world problems in POMDPs and the form of MDPs. In these models, the available opportunities later affect each of the decisions made.

However, we are not concerned about the improvement or development of new Reinforcement Learning algorithms or methods in this work, but rather to search and apply the right algorithm that would be pertinent to find appropriate results and a precise solution for our problems.

We may look for an appropriate choice of approach and resolve the algorithm to be made because Reinforcement Learning Methods are usually completed when solving large and complex problems. Instead of Selecting One algorithm, the IAPTS should depend on different algorithms when solving the complex environment of Penetrating Testing Partially Observable Markov Decision Process (POMDP). It will become simpler and more adaptable that IAPTS should select different algorisms for solving the complex problems in POMDP environments in Penetration testing. To work according to context or available scenario, the IAPTS will change itself to select and use the most appropriate approach to solving the problem. Moreover, the use of one algorithm for problem-solving may make it challenging for IAPTS in terms of available Computational resources, Memory and Time resources, so the choice of selecting different algorithms is judgemental for IAPTS and selecting the right approach by which accuracy is usually immolated to acceptance.

While working with a large number of transactions or choosing a scheme of static reward, it is important to note that large environments will also create big challenges in solving the algorithms. There are two main categories of Solving Algorithms used in Reinforcements learning, including

1. Policy Search Solving
2. Oriented Solving

These are two different approaches to solving the problem of maximizing rewards in a reinforcement learning (RL) environment. The first approach, known as the reward approach, involves developing an optimized reward function that takes into account both immediate and long-term rewards. However, this approach can be time-consuming and complex if the problem representation is not optimized.

The second approach, policy search, focus on constructing a decision policy graph that maximizes the long-term reward. To implement both approaches, the researchers used both on-policy and off-policy implementation methods and used a perfect already designed POMDP-solving approach to evaluate the performance of different solving algorithms. The article concludes by mentioning the shortlisted algorithms that were initially considered.

The reward approach involves developing a comprehensive and optimized reward function that considers both immediate and long-term rewards to determine the best possible rewarding scheme for each transition and observation. However, this approach can be time-consuming and complex, especially if the problem representation is not optimized. [36]

The decision policy graph is constructed by policy search in the other way around and it is done by understanding the policies that would impact and maximize long-term reward and doing internal environment internal mapping.

In this research, both reward-optimization and policy-search approaches were used, along with on-policy and off-policy implementation methods to ensure the best quality policies were found. To evaluate the performance of different solving algorithms, the researchers used a powerful off-the-shelf POMDP-solver and state-of-the-art algorithms, excluding external factors. Several algorithms were initially shortlisted for consideration. [37]

* 1. **PERSEUS Algorithm**

The PERSEUS Algorithm is a ground-breaking approach to solving partially observable Markov decision processes (POMDPs) that offers a more efficient and effective solution to the problem of agent planning under uncertainty. Unlike other point-based methods, PERSEUS performs approximate value backup stages that improve the value of many belief points with a single backup. By selecting a randomly chosen subset of points within the belief set, the algorithm can back up only those points sufficient for improving the value of each belief point in the set. Additionally, PERSEUS performs a policy iteration along with a single back when it searches through the finite state for stochastic space.

This approach makes PERSEUS an exceptionally efficient algorithm for solving large-scale POMDP problems. With its approximate solving nature and the ability to work and perform on a surplus theory set illustrated by simulating decision sequences, PERSEUS shows tremendous potential for accelerating the solving process and achieving optimal results. The algorithm is a breakthrough in POMDP research and offers a new path forward for agent planning under uncertainty.

* 1. **GIP Algorithm**

The GIP algorithm is revolutionizing POMDP exact-solving methods with its reliance on incremental pruning. Instead of using LPs to check for dominating vectors, GIP uses Benders decomposition and only a fraction of the original LP constraints. GIP has been proven to outperform common vector pruning algorithms for POMDPs, reducing running time and memory usage in large POMDP contexts. The latest version of GIP is considered the fastest optimal pruning-based POMDP available.

In exciting news, the GIP algorithm has been updated to allow for external belief sampling, eliminating the need to sample beliefs from the POMDP environment at the beginning of the solving process. This change will enable agents to upload their beliefs directly to the RL environment for more efficient use. Overall, GIP is a game-changer in the world of POMDP solving algorithms, and its latest updates make it even more effective and accessible for users.

The GIP (Generalized Iterative Policy) algorithm is a reinforcement learning algorithm that can be used in Partially Observable Markov Decision Processes (POMDPs). POMDPs are a type of decision-making problem where the agent does not have access to the full state of the environment, but only observes a partial state through sensors. The goal of the agent is to take actions that maximize a long-term reward.

In POMDPs, the optimal policy is a mapping from the current belief state (a probability distribution over the possible states) to actions. The GIP algorithm can be used to learn the optimal policy in POMDPs by iteratively improving an initial policy. The GIP algorithm works by first computing the value function for the current policy. The value function represents the expected discounted reward for each belief state under the current policy. This can be computed using dynamic programming techniques such as the *Bellman equation.* [49]

Once the value function has been computed, the GIP algorithm updates the policy by greedily selecting the action that maximizes the expected reward for the current belief state, based on the value function. This process is repeated iteratively until convergence.

The GIP algorithm is effective in solving POMDPs in a variety of domains, including robotics, autonomous driving, and game-playing. However, it can be computationally expensive and may not scale well to large state spaces. Therefore, researchers continue to explore new algorithms and techniques to address these challenges.

* 1. **PEGASUS Algorithm**

PEGASUS is a cutting-edge policy-search algorithm designed to tackle the challenge of solving large Markov decision processes (MDPs) and partially observable Markov decision processes (POMDPs). This innovative approach, initially proposed by [20], involves transforming any MDP or POMDP into an equivalent POMDP in which all state transitions are deterministic. It decreases the impact of a problem for policy search to the one where consideration is done with transactions with POMDPs only.

To further simplify the policy search process, PEGASUS estimates the value of all policies, allowing for easy identification of high-value policies. This purposeful algorithm has been presented already its capabilities in solving complex problems because it generates a polynomial dependency on horizontal time rather than exponential dependencies. As a result, PEGASUS is an ideal solution for penetration testing POMDP solving and has paved the way for even more advanced policy search algorithms in the future.

Typically, policy search methods require access to a POMDP either by executing trajectories or using a black-box "generative model" that enables the learner to try actions from arbitrary states. In this paper, however, we assume a more powerful model than these - an implementation of a generative model without an internal random number generator. This means the model has to ask us for random numbers whenever it needs them, such as when it needs a source of randomness to draw samples from the POMDP's transition distributions. This simple change results in what we call a deterministic simulative model, which turns out to be surprisingly powerful.

Using this deterministic simulative model, we demonstrate how we can reduce the problem of policy search in any arbitrary POMDP to one in which all the transitions are deterministic. This means that taking an action in a state will always deterministically result in transitioning to some fixed state, except for the initial state which may still be random. This reduction is achieved by changing/reshaping the actual POMDP into a similar one with only calculating transitions.

The policy search algorithm we selected then works on this transformed POMDP, searching for a high-value policy. By assuming access to a deterministic simulative model, we can significantly simplify the policy search problem, which has implications for a range of applications in which POMDPs are used. [48]

* 1. **Other Candidates**

Some other Reinforcement Algorithms like Finite Grid and Backwards Induction along with the addition of the candidates. These RL Algorithms are used to find the shortest path when more than one policy is available and it is a point-based value iteration. POMDP-solver software has already many proposed algorithms included in it, and an emphasized implementation is provided along with its power to improve its version over time.

"Revolutionary algorithm implemented to bridge the gap between theoretical research and real-world industry practices. Highly recommended 'ready solution' utilized for benchmarking purposes, yielding promising results. Ground-breaking Palm leaf search (PLEASE) algorithm evaluated and proven effective in solving complex POMDPs with large observation spaces." [24]. In Reinforcement Learning, a finite grid is a common setting for many problems. A finite grid is a graph of nodes or states where each state represents a specific location in the grid. These states can be thought of as discrete points in space, and an agent in a particular state can take action to move to other adjacent states[38]. The objective of the agent is to learn a policy that maximizes its long-term rewards in this grid environment. Backward Induction is a common algorithm used in Reinforcement Learning to solve finite grid problems. It is a dynamic programming method that works by starting from the last state and calculating the optimal value of each state recursively. The optimal value of a state is the expected sum of rewards that the agent can accumulate by following the optimal policy from that state. The backward induction algorithm starts by initializing the value of the last state in the grid to zero, and then it moves back to the previous state. For each state, the algorithm calculates the optimal value by considering all possible actions that the agent can take from that state and the resulting values of the next states. The optimal value is then stored and used to calculate the optimal value of the previous state. The process continues until the optimal values of all states in the grid are calculated.

Once the optimal values of all states are known, the backward induction algorithm can calculate the optimal policy for the agent. The optimal policy is the set of actions that the agent should take at each state to maximize its long-term rewards. The policy can be determined by selecting the action that leads to the state with the highest optimal value. [39]

In conclusion, the finite grid and backward induction algorithms are essential tools in Reinforcement Learning. They enable agents to learn optimal policies in finite grid environments and can be extended to more complex settings with larger state spaces[47]. These algorithms have practical applications in areas such as robotics, gaming, and finance, where agents need to make optimal decisions in uncertain and dynamic environments.

* 1. **Deep Reinforcement Learning Algorithms**

In recent years, there has been a surge of interest in Deep Reinforcement Learning (DRL) algorithms, which have demonstrated impressive results across various domains. These algorithms operate similarly to traditional reinforcement learning methods, but they incorporate deep neural networks for greater adaptability to environmental changes. One exciting application of DRL is in the field of auto penetration testing, where it has shown promising results. [40]

Researchers have proposed DRL-based auto penetration testing methods using the NAS and Python Tensorflow packages, implementing deep Q-learning agents for testing in NAS environments. While the results in smaller scenarios were encouraging, larger networks have yet to be fully explored.

Another proposed method involved the A3C model, which focused on reducing training time in DRL-based penetration testing by utilizing parallel computation. This approach has shown promise in reducing the overall time required for testing and could be acceptable for post-penetration testing scenarios. [41]

Overall, DRL algorithms offer flexibility, efficiency, and scalability, making them a promising tool for a variety of applications, including auto penetration testing. While more research is needed to fully realize their potential, DRL algorithms are certainly at the forefront of revolutionizing machine learning.

DRL algorithms build on this concept by utilizing deep neural networks to represent the agent's policy, value function, or both. These deep networks can handle high-dimensional inputs, making them suitable for solving problems with complex state spaces, such as image or video data. DRL algorithms have proven to be particularly effective in solving problems where traditional techniques have failed, such as executing complex penetrating testing scenarios. A major advantage of DRL algorithms is their ability to learn from raw sensory inputs. This means that the agent can learn to perceive and understand the environment without any pre-processing or feature extraction. For example, in the context of robotics, DRL algorithms can learn to control robotic arms by directly observing the environment through cameras, without requiring any manual calibration or feature engineering. [42]

Another important feature of DRL algorithms is their ability to learn from sparse rewards. In many real-world problems, the reward signal is only available intermittently or in a delayed manner, making it difficult for traditional techniques to learn. DRL algorithms can learn from such sparse rewards by leveraging the temporal structure of the problem, i.e., by estimating the value of a state or action based on the expected future rewards.

DRL algorithms can be broadly categorized into two types: model-based and model-free. Model-based DRL algorithms involve learning a model of the environment, i.e., a function that predicts the next state and reward given the current state and action. Model-based algorithms can be effective in situations where the environment is well-understood and can be accurately modelled. [46] However, they can be computationally expensive and may not scale well to large or complex environments.

Model-free DRL algorithms, on the other hand, do not require a model of the environment. Instead, they learn the policy or value function directly from the raw sensory inputs. Model-free algorithms can be more efficient and scalable than model-based algorithms, but they may require more data to achieve good performance. [43]

Some of the most popular DRL algorithms include Q-learning, SARSA, Deep Q-Networks (DQNs), Actor-Critic, and Proximal Policy Optimization (PPO). Q-learning and SARSA are traditional reinforcement learning algorithms that extended and enhanced deep neural networks. In video gaming, the DQNs were the first DRL algorithms to achieve human-level performance in video games[44]. Actor-Critic algorithms combine the advantages of policy-based and value-based methods. PPO is a state-of-the-art algorithm that uses a trust region optimization approach to improve stability and convergence. [45]

In conclusion, Deep Reinforcement Learning algorithms have emerged as a powerful tool for solving complex problems in various domains. By combining the concept of reinforcement learning with deep neural networks, DRL algorithms can handle high-dimensional inputs, learn from raw sensory inputs, and learn from sparse rewards. Meanwhile, although good results have been achieved by Deep Reinforcement Learning Algorithms in recent times, there are still many challenges and opportunities for future research, such as improving sample efficiency, developing better exploration strategies, and adapting to changing environments. [51]

Appendix B: Data and Programs Used for this Report[[1]](#footnote-1)

1. Scenario files

- Tiny

# A tiny standard (one public network) network configuration

#

# 3 hosts

# 3 subnets

# 1 service

# 1 process

# 1 os

# 1 exploit

# 1 Privilege escalation

#

# Optimal path:

# (e\_ssh, (1, 0)) -> subnet\_scan -> (e\_ssh, (3, 0)) -> (pe\_tomcat, (3, 0))

#     -> (e\_ssh, (2, 0)) -> (pe\_tomcat, (2, 0))

# Score = 200 - (6\*1) = 194

#

subnets: [1, 1, 1]

topology: [[ 1, 1, 0, 0],

           [ 1, 1, 1, 1],

           [ 0, 1, 1, 1],

           [ 0, 1, 1, 1]]

sensitive\_hosts:

  (2, 0): 100

  (3, 0): 100

os:

  - Linux

services:

  - ssh

processes:

  - tomcat

exploits:

  e\_ssh:

    service: ssh

    os: Linux

    prob: 0.8

    cost: 1

    access: user

privilege\_escalation:

  pe\_tomcat:

    process: tomcat

    os: Linux

  Linuxb: 1.0

    cost: 1

    access: root

service\_scan\_cost: 1

os\_scan\_cost: 1

subnet\_scan\_cost: 1

process\_scan\_cost: 1

host\_configurations:

  (1, 0):

    os: Linux

    services: [ssh]

    processes: [tomcat]

    # which services to deny between individual hosts

    firewall:

      (3, 0): [ssh]

  (2, 0):

    os: Linux

    services: [ssh]

    processes: [tomcat]

    firewall:

      (1, 0): [ssh]

  (3, 0):

    os: Linux

  Linuxvices: [ssh]

    processes: [tomcat]

# two rows for each connection between subnets as defined by topology

# one for each direction of connection

# List which services to allow

firewall:

  (0, 1): [ssh]

  (1, 0): []

  (1, 2): []

  (2, 1): [ssh]

  (1, 3): [ssh]

  (3, 1): [ssh]

  (2, 3): [ssh]

  (3, 2): [ssh]

step\_limit: 1000

* Medium

# A medium standard (one public subnet) network configuration

#

# 16 hosts

# 5 subnets

# 2 OS

# 5 services

# 3 processes

# 5 exploits

# 3 priv esc

#

# |A| = 16 \* (5 + 3 + 4) = 192

#

# Optimal path:

#  (e\_http, (1, 0)) -> subnet\_scan -> (e\_smtp, (2, 0)) -> (pe\_schtask, (2, 0) -> (e\_http, (3, 1))

#      -> subnet\_scan -> (e\_ssh, (5, 0)) -> (e\_samba, (5, 0))

#  Score = 200 - (2+1+3+1+2+1+3+2) = 185

#

subnets: [1, 1, 5, 5, 4]

topology: [[ 1, 1, 0, 0, 0, 0],

           [ 1, 1, 1, 1, 0, 0],

           [ 0, 1, 1, 1, 0, 0],

           [ 0, 1, 1, 1, 1, 1],

           [ 0, 0, 0, 1, 1, 0],

           [ 0, 0, 0, 1, 0, 1]]

sensitive\_hosts:

  (2, 0): 100

  (5, 0): 100

os:

  - Linux

  - windows

services:

  - ssh

  - ftp

  - http

  - samba

  - smtp

processes:

  - tomcat

  - daclsvc

  - schtask

exploits:

  e\_ssh:

    service: ssh

    os: Linux

    prob: 0.9

    cost: 3

    access: user

  e\_ftp:

    service: ftp

    os: windows

    prob: 0.6

    cost: 1

    access: root

  e\_http:

    service: http

    os: None

    prob: 0.9

    cost: 2

    access: user

  e\_samba:

    service: samba

    os: Linux

    prob: 0.3

    cost: 2

    access: root

  e\_smtp:

    service: smtp

    os: windows

    prob: 0.6

    cost: 3

    access: user

privilege\_escalation:

  pe\_tomcat:

    process: tomcat

    os: Linux

    prob: 1.0

    cost: 1

    access: root

  pe\_daclsvc:

    process: daclsvc

    os: windows

    prob: 1.0

    cost: 1

    access: root

  pe\_schtask:

    process: schtask

    os: windows

    prob: 1.0

    cost: 1

    access: root

service\_scan\_cost: 1

os\_scan\_cost: 1

subnet\_scan\_cost: 1

process\_scan\_cost: 1

host\_configurations:

  (1, 0):

    os: Linux

    services: [http]

    processes: []

  (2, 0):

    os: windows

    services: [smtp]

    processes: [schtask]

  (3, 0):

    os: windows

    services: [ftp]

    processes: [schtask]

  (3, 1):

    os: windows

    services: [ftp, http]

    processes: [daclsvc]

  (3, 2):

    os: windows

    services: [ftp]

    processes: []

  (3, 3):

    os: windows

    services: [ftp]

    processes: [schtask]

  (3, 4):

    os: windows

    services: [ftp]

    processes: [schtask]

  (4, 0):

    os: Linux

    services: [ssh]

    processes: []

  (4, 1):

    os: Linux

    services: [ssh]

    processes: []

  (4, 2):

    os: Linux

    services: [ssh]

    processes: []

  (4, 3):

    os: windows

    services: [ssh, ftp]

    processes: [tomcat]

  (4, 4):

    os: windows

    services: [ssh, ftp]

    processes: [tomcat]

  (5, 0):

    os: Linux

    services: [ssh, samba]

    processes: []

  (5, 1):

    os: Linux

    services: [ssh, http]

    processes: [tomcat]

  (5, 2):

    os: Linux

    services: [ssh]

    processes: []

  (5, 3):

    os: Linux

    services: [ssh]

    processes: []

firewall:

  (0, 1): [http]

  (1, 0): []

  (1, 2): [smtp]

  (2, 1): [ssh]

  (1, 3): []

  (3, 1): [ssh]

  (2, 3): [http]

  (3, 2): [smtp]

  (3, 4): [ssh, ftp]

  (4, 3): [ftp, ssh]

  (3, 5): [ssh, ftp]

  (5, 3): [ftp, ssh]

step\_limit: 2000

1. Programs

Table A1|Files and description in the program Environment module

|  |  |
| --- | --- |
| File | Description |
| action.py | Contains action class which defines an action in the NAS |
| environment.py | Contains the main class for the NAS. It controls the network attack model and provides the main function with which to interact with the NAS |
| gym\_env.py | Contains a wrapper around the NASimEnv |
| host\_vector.py | This is the main class for storing and updating the state of a single host |
| network.py | Network model class, which defines and controls the behaviour and configuration of the network in the NAS |
| observation.py | Contains all functionality to generate the observation from the states and actions of the hosts |
| render.py | Contains functionality for rendering the environment and episodes |
| state.py | Contains the state class which defines a state in the NAS |
| utils.py | Contains utilities to get the information for the network such as minimum steps to hosts and minimum subnet depth |

Table A2|Files and description in the Scenarios module

|  |  |
| --- | --- |
| File | Description |
| generator.py | Contains functionality for generating scenarios |
| host.py | Contains functionality to store initial scenario data for a host |
| loader.py | Contains functionality to save the scenario from the file |
| scenario.py | Contains functionality for consistent positioning of host state and obs in state and obs matrices |
| utils.py | Contains utilities to get the name the of scenario file and load it |

Table A3|Files and description in the Agent module

|  |  |
| --- | --- |
| File | Description |
| dqn\_agent.py | Contains Deep-Q learning agent class |

Table A4|Files and description for the Experiment files

|  |  |
| --- | --- |
| File | Description |
| dqn\_train.py | Contains functionality to train the DQN agent |
| dqn\_train.py | Contains functionality to test the DQN agent |

Table A5|Files and description for scenarios

|  |  |
| --- | --- |
| File |  |
| tiny. yaml | Benchmark scenario file for the tiny network |
| medium. yaml | Benchmark scenario file for the medium network |

References[[2]](#footnote-2)

[1] J.Schwartz, Autonomous Penetration Testing using Reinforcement Learning, School of Information Technology and Electrical Engineering, University of Queensland

[2] Mohamed C.GHANEM and Thomas M.CHEN Reinforcement Learning for Efficient Network Penetration Testing, <https://www.researchgate.net/publication/330590120>

[3] Aileen G.Bacudio, Xiaohong Yuan, Bei-Tseng Bill Chu and Monique Jones, An Overview of penetration testing, International Journal of Security & Its Applications (IJNSA), Vol.3, No.6, November 2011

[4] C. Heinl, Artificial (intelligent) agents and active cyber defence: policy implications. 6th Int. Confe. on Cyber Conflict. NATO CCD COE Publications, 2016, Tallinn.

[5] M. Backes, J. Hoffmann, R. Kunnemann, P. Speicher and M. Steinmetz, Simulated penetration testing and mitigation analysis. http://arxiv.org/abs/1705.05088, 2017.

[6] Alessandro Confido, Evridiki N.Ntagiou, Marcus Wallum, Reinforcing Penetration Testing Using AI. 2022 IEEE Aerospace conference, 2022.

[7] Y. Andrew and M. Jordan, PEGASUS: A policy search method for large MDPs and POMDPs. 16th Conf. on Uncertainty in Artificial Intel., 2013.

[8] T. Schaul, J. Quan, I. Antonoglou and D. Silver, Prioritized experience replay, Google DeepMind. ICLR 2016.

[9] K. Veeramachaneni, I. Arnaldo, A. Cuesta-Infante, V. Korrapati, C. Bassias and K. Li, AI2: Training a big data machine to defend. CSAIL, MIT Cambridge, 2016.

[10] K. Durkota, V. Lisy, B. Bosansk and C. Kiekintveld, Optimal network security hardening using attack graph games. 24th Int. Joint Conf. on Artificial Intelligence (IJCAI-2015), 2015.

[11] N. Meuleau, K. Kim, L. Kaelbling and A. Cassandra, Solving POMDPs by searching the space of finite policies. 15th Conf. on Uncertainty in Artificial Intel., 2013.

[12] NIST, Computer Security Resource Center - National Vulnerability Database, https://nvd.nist.gov, 2017.

[13]. Creasey, J.; Glover, I. A guide for Running an Effective Penetration Testing Program; CREST Publication: Slough, UK, 2017. Available online: http://www.crest-approved.org (accessed on 18 December 2019).

[14]. Almubairik, N.; Wills, G. Automated penetration testing based on a threat model. In Proceedings of the 11th International Conference for Internet Technologies and Secured Transactions, ICITST, Barcelona, Spain, 5–7 December 201.

[15]. Applebaum, A.; Miller, D.; Strom, B.; Korban, C.; Wol, R. Intelligent, automated red team emulation. In Proceedings of the 32nd Annual Conference on Computer Security Applications (ACSAC’16), Los Angeles, CA, USA, 5–8 December 2016; pp. 363–373.

[16]. Obes, J.; Richarte, G.; Sarraute, C. Attack planning in the real world. arXiv 2013, arXiv:1306.4044.

[17]. Spaan, M. Partially Observable Markov Decision Processes, Reinforcement Learning: State of the Art; Springer: Berlin/Heidelberg, Germany, 2012.

[18]. Hoffmann, J. Simulated penetration testing: From Dijkstra to aaTuring Test++. In Proceedings of the 25th International Conference on Automated Planning and Scheduling, Israel, 7–11 June 2015.

[19]. Sarraute, C. Automated attack planning. Available online: https://arxiv.org/abs/1307.7808 (accessed on 20 December 2019).

[20]. Qiu, X.; Jia, Q.; Wang, S.; Xia, C.; Shuang, L. Automatic generation algorithm of penetration graph in penetration testing. In Proceedings of the Ninth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, Guangdong, China, 8–10 November 2014.

[21]. Heinl, C. Artiﬁcial (intelligent) agents and active cyber defence: Policy implications. In Proceedings of the 6th International Conference On Cyber Conﬂict (CyCon 2014), Tallinn, Estonia, 3–6 June 2014.

[22]. Sarraute, C.; Buffet, O.; Hoffmann, J. POMDPs make better hackers: Accounting for uncertainty in penetration testing. Available online: https://arxiv.org/abs/1307.8182 (accessed on 20 December 2019).

[23]. Backes, M.; Hoffmann, J.; Kunnemann, R.; Speicher, P.; Steinmetz, M. Simulated Penetration Testing and Mitigation Analysis. arXiv 2017, arXiv:1705.05088.

[24]. Ghanem, M.; Chen, T. Reinforcement Learning for Intelligent Penetration Testing. In Proceedings of the WS4 the World Conference on Smart Trends in Systems, Security and Sustainability, London, UK, 30–31 October 2018.

[25]. J. Creasey, and I. Glover, A guide for running an effective Penetration Testing program,721 http://www.crest-approved.org. CREST Publication, 2017.722

[26]. N. Almubairik, G. Wills, Automated penetration testing based on a threat model. 11th International723 Conference for Internet Technologies and Secured Transactions, ICITST, 2016.724

[27]. A. Applebaum, D. Miller, B. Strom, C. Korban, and R. Wol, Intelligent, automated red team emulation. 32nd725 Annual Conference on Computer Security Applications (ACSAC ’16), 2016, pp. 363-373.726

[28]. J. Obes, G. Richarte, and C. Sarraute, Attack planning in the real world. Journal CoRR Article, 2013,727 abs/1306.4044.728

[29]. M. Spaan, Partially Observable Markov Decision Processes, Reinforcement Learning: State of the Art,729 Springer Verlag, 2012.730

[3J. Hoffman, Simulated penetration testing: From Dijkstra to a Turing Test++. 25thInt. Conf. onAutomated731 Planning and Scheduling, 2015, AAAI Press.732

[31] C.Sarraute, Automated attack planning. Instituto Tecnologicode Buenos-aires, Ph.D. Thesis,2012, Argentina.733

[32]. X. Qiu, Q. Jia, S. Wang, C. Xia and L. Shuang, Automatic generation algorithm of penetration graph in734 penetration testing, 19th Int. Conf. on P2P, Parallel, Grid, Cloud and Internet Computing, 2014.735

[33]. C. Sarraute, O. Buffet and J. Hoffmann, POMDPs make better hackers: Accounting for uncertainty in736 penetration testing. 26th AAAI Conf. on Artiﬁcial Intel. (AAAI’12), pp. 1816–1824, July 2012.737

[34]. C. Heinl, Artiﬁcial (intelligent) agents and active cyber defence: policy implications. 6th Int. Confe. on738 Cyber Conﬂict. NATO CCD COE Publications, 2016, Tallinn.739

[35]. M. Backes, J. Hoffmann, R. Kunnemann, P. Speicher and M. Steinmetz, Simulated penetration testing and740 mitigation analysis. http://arxiv.org/abs/1705.05088, 2017.741

[36]. S. Jimenez, T. De-la-rosa, S. Fernandez, F. Fernandez and D. Borrajo, A review of machine learning for742 automated planning. The Knowledge Engineering Review, Vol. 00:0, pp. 1–24. 2009.743

[37]. Y. Andrew and M. Jordan, PEGASUS: A policy search method for large MDPs and POMDPs. 16th Conf. on744 Uncertainty in Artiﬁcial Intel., 2013.745

[38]. T. Schaul, J. Quan, I. Antonoglou and D. Silver, Prioritized experience replay, Google DeepMind. ICLR 2016.746

[39]. K. Veeramachaneni, I.Arnaldo, A.Cuesta-Infante, V.Korrapati, C.BassiasandK.Li, AI2: Trainingabigdata747 machine to defend. CSAIL, MIT Cambridge, 2016.748

[40]. K. Durkota, V. Lisy, B. Bosansk and C. Kiekintveld, Optimal network security hardening using attack graph749 games. 24th Int. Joint Conf. on Artiﬁcial Intelligence (IJCAI-2015), 2015.750

[41]. N. Meuleau, K. Kim, L. Kaelbling and A. Cassandra, Solving POMDPs by searching the space of ﬁnite751 policies. 15th Conf. on Uncertainty in Artiﬁcial Intel., 2013.752

[42]. NIST, Computer Security Resource Center - National Vulnerability Database, https://nvd.nist.gov, 2018.753

[43]. E. Walraven, amd M. Spaan. Accelerated Vector Pruning for Optimal POMDP Solvers, Proceedings of the754 Thirty-First AAAI Conference on Artiﬁcial Intelligence, 2017.755

[44]. M. Ghanem, and T. Chen. Reinforcement Learning for Intelligent Penetration Testing. World Conference on756 Smart Trends in Systems, Security and Sustainability. 2018.

[45] Symantec Corporation, “Internet Security Threat Report,” Volume 22, Apr. 2017.

[46] Australian Cyber Security Centre, “ACSC Threat Report,” Australian Government, 2017.

[47] A. IVan, “Why Attacking Systems Is a Good Idea,” ​IEEE Secur. Priv. ​ , 2004.

[48] R. S. Sutton and A. G. Barto, “Reinforcement learning: An introduction,” 2011.

[49] B. Arkin, S. Stender, and G. McGraw, “Software penetration testing,” ​IEEE Secur. Priv. ​ , vol. 3, no. 1, pp. 84–87, Jan.-Feb 2005.

[50]Hansen, E. A., and Feng, Z. 2000. Dynamic programming for POMDPs using a factored state representation. In Proceedings of the 5th international conference on Artificial Intelligence Planning & Scheduling.

[51]Cassandra, A.; Littman, M.; and Zhang, N. 1997. Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes. In Proceedings of the 13th Annual Conf. on Uncertainty in Artificial Intelligence (UAI-97), 54–61.

into in Artificial Intelligence (UAI-97), 54–61.

1. Amend headings and text as needed, remove this footnote [↑](#footnote-ref-1)
2. It is highly recommended that you use EndNote to make the list of references for this document, remove this footnote [↑](#footnote-ref-2)