

# The Proposal of Double Agent Architecture using Actor-critic Algorithm for Penetration Testing

Hoang Viet Nguyen<sup>1</sup>, Songpon Teerakanok<sup>2</sup> <sup>a</sup>, Atsuo Inomata<sup>3</sup> and Tetsutaro Uehara<sup>1</sup> <sup>b</sup>

<sup>1</sup>*Cyber Security Lab, College of Information Science and Engineering, Ritsumeikan University, Japan*

<sup>2</sup>*Research Organization of Science and Technology, Ritsumeikan University, Japan*

<sup>3</sup>*Graduate School of Information Science and Technology, Osaka University, Japan*

**Keywords:** Penetration Testing, Deep Reinforcement Learning.

**Abstract:** Reinforcement learning (RL) is a widely used machine learning method for optimal decision-making compared to rule-based methods. Because of that advantage, RL has also recently been used a lot in penetration testing (PT) problems to assist in planning and deploying cyber attacks. Although the complexity and size of networks keep increasing vastly every day, RL is currently applied only for small scale networks. This paper proposes a double agent architecture (DAA) approach that is able to drastically increase the size of the network which can be solved with RL. This work also examines the effectiveness of using current popular deep reinforcement learning algorithms including DQN, DDQN, Dueling DQN and D3QN algorithms for PT. The A2C algorithm using Wolpertinger architecture is also adopted as a baseline for comparing the results of the methods. All algorithms are evaluated using a proposed network simulator which is constructed as a Markov decision process (MDP). Our results demonstrate that DAA with A2C algorithm far outweighs other approaches when dealing with large network environments reaching up to 1000 hosts.

## 1 INTRODUCTION


Nowadays, the rapid development of network infrastructure has been increasing the risk of network vulnerability. Penetration testing (PT) that is being used widely in many systems has been seen as an important approach in the development of network protection. The main purpose of the approach is protecting the system from potential attacks by simulating real-life hacker's activities trying to find and attack all the existing security vulnerabilities on the network.


However, there are still many things that prevent organizations from accessing the advantages of PT. PT is often considered an expensive approach, performing it consumes a lot of resources including manpower. The operator who has a wide and deep knowledge of security is required to perform PT. Using automatic PT instead of depending on human resources is a popular approach to solve these problems. Currently, many frameworks and tools supporting PT have been developed such as Metasploit, Nessus and Tenable. Unfortunately, these tools usually only sup-

port security experts in scanning vulnerabilities rather than assist them exploitation.

In recent years, machine learning (ML) is opening up a new approach in effectively solving complex problems. ML has been proven that it is able to deal with difficult problems more quickly and accurately than humans in some cases (Chui et al., 2016). Reinforcement learning (RL) a type of ML has received great attention lately. RL is able to make a sequence decision to interact with an unstable environment in order to get the best reward out of it (Kaelbling et al., 1996). Unlike supervised learning, the RL agent finds the good way to adapt to the environment by itself without needing pre-labelled data.

The ability of RL shows that it is perfectly suited to be used to solve PT problems. Leveraging the power of RL in unstable environments is a good approach to conduct PT while the RL agent can automatically decide the action needed to perform. In fact, there are a few researchers who pursued this direction and have achieved some promising results. Schwartz et al. showed that RL algorithms are able to search for the good root of exploiting network and find the attack policy that successfully attacks all target machines (Schwartz and Kurniawati, 2019). Ghanem et al., on

<sup>a</sup>  <https://orcid.org/0000-0002-1058-149X>

<sup>b</sup>  <https://orcid.org/0000-0002-8233-130X>

the other hand, proposed the ability to combine RL with existing PT frameworks to perform tasks with no human interference (Ghanem and Chen, 2020).

Despite achieving many promising results, all above-mentioned approaches, unfortunately, have a fundamental limitation in the size of the environment that can be solved. To be clear, this limitation does not only appear when using RL with PT but also is a general problem of RL. Schwartz et al. showed that their algorithm performance rapidly decreases when the network has more than 43 machines (Schwartz and Kurniawati, 2019). IAPTS, a POMDP approach, tried to improve the efficiency on large networks, is tested with a maximum of 100 machines (Ghanem and Chen, 2020). Using a multi-level system, 4AL, a promising model-based method, maintains its performance when the size of the environment reaches 100 machines and 100 services (Sarraute et al., 2013).

In this paper, we propose a double agent architecture (DAA) which use a popular RL algorithm called the advantage actor-critic algorithm (A2C) (Mnih et al., 2016) to conduct PT in large networks. DAA uses model to attack environments in term of Markov decision processes (MDP). The aim of DAA is to be able to do PT on a large network system having around 1000 machines by remains its performance. DAA utilizes a network simulator that is built by inheriting and developing the features of the network attack simulator (NAS) (Schwartz and Kurniawati, 2019). The network simulator is then translated into an MDP problem to be determined by agents. The A2C algorithm using Wolpertinger architecture (WA) with multiple level action embedding is also adopted as a baseline for comparing the results (Hoang et al., 2019).

The main contributions in this paper are summed up as follows:

- We build a JSON format network simulator that is implemented based on the NAS. To add practicality and complexity to the network simulator, there is a few extension rules have been proposed.
- Testing the effectiveness of some popular deep reinforcement learning (DRL) algorithms such as DQN, DDQN, Dueling DQN and D3QN when using in PT.
- A robust DRL architecture has been proposed called double agent architecture (DAA) leveraging the power of the actor-critic algorithm and multi-agent architecture to conduct PT in the large environment but maintain the performance. DAA contains two agents, while the first one is used to learn about the network structure of a network, the second one helps to choose the appropriate exploitable service for the machine selected.

- We have conducted experiments on different network size with various attack scenarios. The impact of size on the performance of the algorithms are compared. The results showed that DQN family algorithms and WA can solve the network having less than 30 and 256 machines respectively, while the DAA works for around 1000 hosts and still remain the performance 81%. The result of DAA, which is superior to other algorithms, is a testament to DAA's ability to perform PT in large networks.

This paper is organized as follows. Section 2 introduces the literature review about related works. The design of the network simulator is described in Section 3. In Section 4, the detailed structure of the DAA is discussed. This section also discusses the implementation of Wolpertinger architecture using for comparison. Section 5 reports conducted experiments and analyze the results. Finally, the last section summaries this study concludes.

## 2 LITERATURE REVIEW

### 2.1 Reinforcement Learning Algorithms

RL approach tries to maximizing the cumulative reward from the environment to find a good plan (Kaelbling et al., 1996). At first, RL is usually seen in game theory and information theory. However, in early years, applying RL method is considered unsuccessful, until Mnih et al. proposed the method named DQN that combine deep neural network with Q-learning (Mnih et al., 2013). The proposed method became a huge success and its results demonstrate that RL approach can be used effectively on real-world problems. Q-learning has been improved by many researchers ever since. Many techniques such as experience replay and double Q-learning are adopted to improve the performance of RL. RL algorithms commonly used today can be mentioned as DQN, DDQN (Hasselt, 2010), Dueling DQN (Dueling DQN) (Wang et al., 2016) and Duelling Double Deep Q Network (D3QN).

Although previous RL approaches have been proven to be effective, it is still limited to fairly low-dimensional problems and lacks scalability. This problem is considered a big challenge for researchers working in the field. Wolpertinger architecture (WA) using actor-critic algorithm tried to solve the environment having large action space by adding action embedding and kNN layers (Dulac-Arnold et al., 2015). The idea of WA is by finding the max-values action

from similar action set, the agent can be able to find the best one. In the previous work, we proposed a multiple level action embedding that can accurately represent the relationship between actions in the RL action space (Hoang et al., 2019). This approach can be used to apply WA for solving PT problems.

## 2.2 Automatic Penetration Testing

Many automatic PT methods have been developed. These methods often focus on automatic planning. At first, the research considered PT as a decision-making problem and used graph methods like decision tree or attack graph (Phillips and Swiler, 1998). Sarraute et al. attempt to solve such issue by using multi-level architecture called 4AL (Sarraute et al., 2013). It divides the PT process into four levels: decomposing the network, attacking components, subnetworks and individual machines. The system built as a partially observable Markov decision process (POMDP) has proven to be useful to a large number of machines and exploits. Unfortunately, the number of exploits and machines tested in this article just stops at around 100.

In recent years, scientists have paid more attention to using RL to apply to PT automation. Schwartz et al. built network attack simulator (NAS) and uses tabular Q-learning and DQN algorithms to solve such problems (Schwartz and Kurniawati, 2019). The experiments of this article which apply to many different network architectures assert that RL is completely capable of solving PTs. However, the article also points out that with networks with too many machines or services, the performance of these algorithms will decrease significantly, even unsolvable.

Ghanem et al. proposed a system called IAPTS (Ghanem and Chen, 2020). This system uses the RL to make attack decisions and uses PT frameworks such as Metasploit to attack. The system also uses expert validation to monitor the output of the system to help the agent be able to make more accurate decisions. Unlike the above paper, the PT of the paper is represented in the form of POMDP and uses the algorithm PEGASUS to solve. However, like the previous articles, this article is only used on medium and small environments (from 10-100 machines). The number of services used is not mentioned in the article.

## 3 THE NETWORK SIMULATOR

This section contains two main parts: an introduction to the network simulator used to perform penetration testing and a description of how to convert

network simulator information into MDP format. In the first part, the paper proposes a network simulator built based on NAS (Schwartz and Kurniawati, 2019), however, some new rules added to improve the complexity of the simulator compared to the previous one. Second, representing in MDP format is necessary for PT problems to be solved by RL algorithms. Further explanation of how to convert PT problems into the right format to use as an input of RL will be discussed in section 3.2.

### 3.1 The Network Model

The network built based on NAS (Schwartz and Kurniawati, 2019) contains basic components such as subnets, connections, hosts, services and firewalls. These components are defined in JSON format and are loaded when needed. This makes the network simulator can be deployed fast and be able to run in different systems. While our connections, firewalls and services contain changes in format and structure, there are also some significant additional rules applied in subnets and hosts. The purpose of these rules is to make the attack process on the PT environment more difficult for attackers comparing to NAS.

#### 3.1.1 Subnets

Sub-networks (subnets) are the basic components making up each network. By default, every host in a subnet is directly connected and can communicate with each other. Subnet address and subnet mask are used to define a subnet. While the address identifies which host belongs to which subnet, the maximum number of machines in each network is determined by the subnet-mask. Hosts on a different subnet can not communicate with each other. The network connections and firewalls control communication between subnets.

When deploying a network attack on our environment, there are several rules that attackers must follow:

- A subnet is considered reachable to attackers if they have successfully attacked at least one host on the subnet which directly connect to it.
- Attackers, at first, do not know how the subnets are connected.
- Attackers, by default, know the number of subnets but do not know the number of machines in each subnet, even if the subnet is currently reachable to the attacker. To be able to gather this information, attackers have to scan the subnet first.
- If the subnet is unreachable, scanning it do not gain any information to attackers.

### 3.1.2 Hosts

A host represents a device that existed in a network and runs certain services. IP addresses are an important component to differentiate hosts from each other. In the network simulator, there are two types of hosts: normal hosts and sensitive hosts. Sensitive hosts are hosts that contain important information that is needed to be protected from adversaries.

Services that are running on each host can be communicated with other hosts in the same or neighbour subnet. Each host can run different services because, in reality, not all machines have the same configuration or running services. Services are seen as vulnerabilities that attackers can exploit using appropriate action. Each service is defined by an ID and two additional information, a score and a cost. The service score represents the probability of the exploitation of service success. For example, if the score of a service is set to 100, then attacks on this service are always successful. The cost of the service represents how difficult to perform an attack on that service.

To exploit a host in the network simulator, the attackers must also comply with the following rules

- Any action on the host takes effect only when the host is contained in a scanned subnet
- Attackers, at first, do not know which services are running in the hosts. This information can be gained after the host is scanned.
- Attackers can exploit a host when having information about which service is running in the hosts.
- Exploitation on the host is only counted as successful when it is hacked into the service running on the host. Even that, it is still possible that the attack fails to depend on the score of the service.

## 3.2 Representing Penetration Testing as Reinforcement Learning Problems

In reinforcement learning, all problems need to be represented in a proper form before it can be solved. It depends on the designer’s intentions in training the agent. In this paper, the network pentesting problem is modelled as an MDP defined by  $\{\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}\}$ . States ( $\mathcal{S}$ ) are defined as full knowledge about the current network. Actions ( $\mathcal{A}$ ) present the possible actions can be performed for each host and subnet on the network. The reward function ( $\mathcal{R}$ ) calculates how good the attacker exploit the network based on certain rules. The result of performing each action is determined by the transition function ( $\mathcal{T}$ ).

### 3.2.1 State

A state,  $s \in \mathcal{S}$  is defined as the knowledge of all hosts existed on the network. Particularly, the state is a vector including information of all hosts. This information is the host state including attacked state, reachable state and which service is running on it. To be clear, each bit in a state vector can belong to one of three types: reachable state, attacked state and services state. First, for the reachable state, a bit can be assigned these value reachable, unreachable or unknown. Second, if a bit belongs to the attacked state, it can receive attacked, not attacked or unknown value. Finally, if a bit represents for a state of a service on a host, it can have one of three values available, unavailable and unknown.

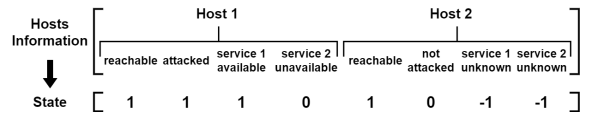


Figure 1: Defining the state space from network information.

Although the information about the subnet is not contained in the state, it plays a crucial role in determining the current state of hosts. A subnet can receive unknown, reachable, and scanned states. The subnet is given an unknown state when an attacker does not know the existence of that subnet. The subnet receives reachable and scanned states following the set of rules applied on subnets mentioned above.

The state of a subnet has a direct influence on the state of hosts in it. A host is considered unknown when the subnet has not been scanned because at this point the attacker does not know any information about the host. A host is considered reachable when it is contained in a scanned subnet. Finally, a host receives an attacked state if an exploit has successfully been used on it.

The state-space includes all possible states of each host including the knowledge about services running on it. Therefore, if the number of machines and services on the network increase, the state space will grow vastly. The equation 1 shows the state space size,  $|\mathcal{S}|$ , where E is the number of exploitable services and H is the number of hosts in the network. The number 2 represents information about the reachable and attacked state of a host. Because every bit in a state vector can receive three possible values, the state space is the exponent of 3.

$$|\mathcal{S}| = 3^{(E+2) \times H} \quad (1)$$

For the equation 1, there are some elements that worthwhile to consider. Although each host has a dif-

ferent number of running services, the number of service state is the same. Whether a service runs on a host or not, it can still be assigned one of three values "available", "unavailable" or "unknown", corresponding to 1, 0 and -1. Therefore, in a state vector, each host is always represented by a number equals to  $(E + 2)$  bits.

### 3.2.2 Action

The action space  $\mathcal{A}$  is defined as the set of all actions within the network simulator. There are three types of action including scan subnets, scan hosts and exploit services. The scan subnets action includes a single scan action for each subnet, while the scan hosts action is scanning for each host. The exploit services action has an exploit for each service and each host on the network.

The scan hosts action simulates the Nmap scanning. It gives users information about current running services and its related information such as open ports, service name and service version. Meanwhile, the scan subnets action is related to host discovery giving all IP address of hosts running on the subnet.

Every service on the network has a matching exploit action. The target host will be counted as attacked if an exploit action successfully conducts on it. Determining the success of any exploit action is checking the set of conditions such as the reachable state of the target host, the availability of the service based on the firewall rules and the action success probability.

The number of actions  $|\mathcal{A}|$  is the total number of three types of actions: scan subnet, scan hosts and exploit services. It is calculated by the number of subnet  $S$ , the number of host  $H$  and the number of exploitable service  $E$  as follows:

$$|\mathcal{A}| = S + H + H \times E \quad (2)$$

Each action also has a cost which represents any real metric such as the time, monetary cost and skill needed.

### 3.2.3 Transition

The transition function,  $\mathcal{T}$ , shows how the environment reacts with performed actions. The next state is determined based on the success of the action. The change in the next state depends on several factors such as the reachable ability of the action target and what type of action being used. These factors are strongly related to the set of rules defined in the network simulator (section 3.1)

### 3.2.4 Reward Function

In any RL problem, the reward is always a crucial component. It is a compass that instructs the agent whether the action taken in the current state is correct. The reward is defined by the function  $\mathcal{R}(s, a, s')$ , from state  $s$  agent take an action  $a$  and finally receive state  $s'$  as the result.

The reward for transitions is the value of any newly successful attacked host minus the cost of the selected action  $a$ . For any action that changes the state of the environment, the little amount of reward  $r'$  is added to encourage the agent. When the agent successfully attacks a normal host, the reward  $r$  will be given to the agent. In the network simulator, 10% of hosts are sensitive hosts containing important information. When successfully attacking these hosts, the reward will be  $(H_{normal}/H_{sensitive}) \times r$ . This formula ensures that the total reward received if attacking sensitive hosts will be equal to the total reward received when attacking normal hosts. This strikes a balance between the agent's attempting to successfully attack sensitive hosts and expanding the reach area by attacking normal hosts.

## 4 THE DOUBLE AGENT ARCHITECTURE (DAA)

### 4.1 Introduction

All reinforcement learning architectures contain two main components an environment and an agent. The basic architecture often has only one agent which directly interacts with the environment. The environment represents the problem that needs to be solved. The agent which has a core of RL algorithms is responsible for finding out the best solution for the problems in the environment. To do that, the agent continuously interacts with the environment by using appropriate actions. At first, the selected actions are not optimized, but based on the feedback information (including new states and rewards) sent by the environment, the agent updates its policy and gradually makes more accurate choices.

When applying RL to PT problems, these properties and behaviour of the agent and the environment are retained. Previous articles have shown that, in this way, RL algorithms is still capable of searching for the good root of successful exploiting target machines on a network (Schwartz and Kurniawati, 2019) (Ghanem and Chen, 2020). However, when the number of hosts and services is too large, its complexity

will increase exponentially. This leads to the situation that the agent is no longer able to solve the problem. Sarraute et al. proposed 4AL algorithm, a domain-specific algorithm, using a multi-level system splitting the PT process into four levels including decomposing the network, attacking components, attacking subnetworks, and attacking individual machines (Sarraute et al., 2013). The results of the paper show that this concept is capable of scaling up performance in solving PT in large environments (tested for up to 100 machines).

Inspired by the concept of dividing the PT process, this paper proposes the double agent architecture (DAA) that divides the PT problem into two steps including understanding the network structure (network topology) and choosing an appropriate exploitable service to attack a certain host. Applying to RL, instead of using only one agent, DAA uses two separate agents, each agent undertakes a different step. This approach is possible to reduce significantly the action space and the state space of PT problem that each agent must solve. Thus, the DAA can be applied to problems with a large number of hosts and services.

## 4.2 Design

As mentioned in 4.1, the state space size and the action space size depend greatly on the number of hosts, subnets and services. This leads to the situation that RL algorithms no longer have the ability to learn to solve problems in such a large environment. In order to solve these problems, this paper proposes an idea that uses two separate agents for learning the network structure and choosing an appropriate service to exploit a certain host. The structuring agent is responsible for learning about the network structure including subnets, hosts, firewalls and the connections between subnets. Meanwhile, the exploiting agent is responsible for picking out the service that is most likely to be a successful attack in a certain host.

The coordinating of two agents is shown on the figure 2. At first, the structuring agent observes the environment and gets a state  $s_1$  from this observation. In the case the agent thinks that the received state still lacks structural information, i.e. the agent thinks it can explore the network more deeply, it will choose structural discovery actions including hosts and subnets scanning. When executing these commands, the agent receives an immediate reward  $r_1$  from the environment and uses it to evaluate the correctness of the action. In the other case, if the agent considers that it is acceptable to collect information or exploit services of a certain host, because of the lacks of service

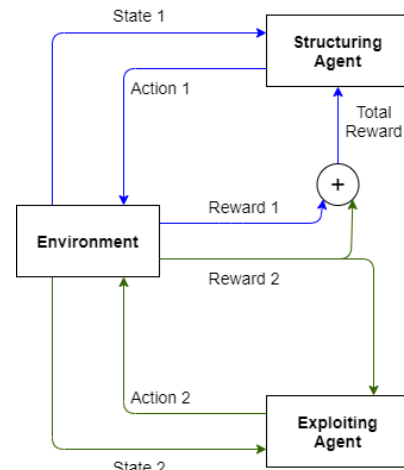


Figure 2: Double agent architecture.

information, it will not directly make decisions but do the action that will trigger the exploiting agent.

The exploiting agent once enabled, will use the state of the selected host as its input (state 2). The selected host's state information will now include the reachable, attacked state and information about the services running on it. At this point, if the agent considers that the information is not enough, it can continue to execute the scan hosts command, otherwise, it can exploit the host with the appropriate service. After the exploiting agent action, the environment will give the agent an immediate reward  $r_2$ . The exploiting agent uses this reward to update its policy. The structuring agent is also aware of this reward and uses the sum of  $r_1$  and  $r_2$  as the result of its action that triggers the exploiting agent. If the action that the structuring agent chooses is scan hosts or scan subnets then  $r_2$  is considered 0. Finally, the structuring agent uses the total reward to update its policy.

The use of the double agent not only reduces significantly the action space and the state space of PT but also increases the agent's learning speed. In addition, separating into two agents also helps to reuse the agents. For example, when the number of services in the network changes, we only need to train the exploiting agent individually. On the other hand, when there is a change in the network structure but there is no change in the number of services, it is completely possible to train the chosen network separately without needed to retrain the exploiting one.

## 4.3 The Environment for Double Agent Approach

The first issue we need to consider when adopting this approach is needed changes in the built environment

to fit into the new architecture. However, this change should be as little as possible to avoid the case that it is too different from the original environment and can be viewed as different problems. Therefore, this paper only proposes very small changes to the current environment so that the environment can be used by both the original method and the double agent method.

#### 4.3.1 State

The state space has not been changed when using DAA. The only difference is that we have to filter the matching state space information as input to agents. The state of the structuring agent will include the reachable and attacked state of all hosts on the network. From the perspective of the pentesting operator, we totally know the total number of hosts currently on the network so this filtering can be done easily. The size of the state space of the first agent is

$$|\mathcal{S}_1| = 3^{2H} \quad (3)$$

On the other hand, the exploiting agent's state will be details of the selected host including reachable, attacked state and all information about that exploitable services. The state space size of the exploiting agent is

$$|\mathcal{S}_2| = 3^E \quad (4)$$

With this size, agents can completely solve the problem even though the number of hosts and services increases. The complexity is now much smaller than before

#### 4.3.2 Action

The action space of the environment as mentioned above includes three types: scan subnets, scan hosts and exploit services. The selection of scan actions, including scan hosts and scan subnets, can be determined by the structuring agent to retrieve information about the network structure. However, this agent cannot decide whether to exploit service or which service should be used to exploit the machine. Therefore, an extension action named choose hosts is added to this agent. This action is chosen when the structuring agent think that there is a host that is likely successfully attacked but the agent can not handle such information. Instead, it sends the current host information to the exploiting agent and let this agent take responsibilities.

With the information received from the structuring agent, the exploiting agent is able to decide which exploit service is available on the host. At this point, it can choose between exploiting the host using such

service or scanning the host to retrieve more information.

With this approach, the structuring agent can choose actions including scan subnets, scan hosts and choose hosts, meanwhile, the exploiting agent can choose actions which works on a certain host such as scan hosts and exploit services on a host. Thus, the action space size of agents has been significantly reduced as follow.

$$|\mathcal{A}_1| = S + 2H \quad (5)$$

$$|\mathcal{A}_2| = H + E \quad (6)$$

#### 4.3.3 Reward Function

Although the reward function does not change, there is a little bit different when using this reward to train agents. For the exploiting agent, the agent still receives the immediate reward in interacting with the environment. For the structuring agent, the two actions scan subnet and scan host still receive an immediate reward. However, the new action, choose host, is a little different. The results of the choose host action are available only after we have finished running the exploiting agent. In other words, the result of choose host will be the sum of the immediate reward of the structuring agent and the exploiting agent.

#### 4.4 The Neural Network of Agents

Neural networks used for agents are built based on actor-critic algorithms. This article uses the advantage actor-critic algorithm (A2C) (Mnih et al., 2016) for its efficiency and ease of implementation. In general, both agents have a relatively similar neural network structure in terms of the number of hidden layers and learning parameters. The required parameters for the A2C algorithm have been constructed according to the previous article (Mnih et al., 2016). However, there is a slight change in the number of neurons per hidden layer, because of the difference in the input size. The actor and critic networks of each agent share the same configuration.

To determine the most suitable hyper-parameters value for neural networks, a tuning process has been performed. The table 1 describes more clearly parameters used and the differences in the configuration of neural networks.

### 5 EXPERIMENTS AND RESULTS

To test the performance of the proposed approach, two experiments will be conducted. The first ex-

Table 1: Agent’s network parameters.

Parameter	Value
No. hidden layers (structuring agent)	3
No. neurons per hidden layer (structuring agent)	512
No. hidden layers (exploiting agent)	3
No. neurons per hidden layer (exploiting agent)	50
Learning rate	$5 \times 10^{-3}$
Discount factor	0.99
Entropy coefficient	$5 \times 10^{-3}$

periment is used to evaluate the results of DAA, the DQN family algorithms and the A2C algorithm using Wolpertinger architecture with multiple level action embedding (Hoang et al., 2019) in the case of small networks. The second experiment is to compare the results between Wolpertinger architecture (WA) and DAA. Metrics are used to evaluate the results of experiments including the reward gained during training and the proportion of sensitive hosts being successfully attacked.

These experiments are conducted to answer the following questions:

- Can DQN family algorithms such as DDQN, Dueling DQN and D3QN as well as A2C algorithm using WA and DAA be used to solve PT problems?
- Does apply a DAA increase problem-solving performance in large environments, especially when compared to WA?
- What is the maximum size of the network that these algorithms can handle?

The experiments use the auto-generated scenarios with the number of hosts ranging from 5 to 1024 and the number of services from 2 to 100. Scenario configurations contain all components and follow the rules of the network simulator in section 3.1. Each host in a scenario contains a different number of running services. For each service, the score value is randomized from 40-90%.

An Ubuntu computer using NVIDIA GeForce RTX 2080 Ti 11GB GPU and AMD Ryzen 9 3900X CPU is used to conduct these experiments

### 5.1 The Experiment Conducted on Small Network Environments

In this test, we designed and implemented several networks on different sizes. The number of hosts (H) on each network is ranging from 5 to 50 hosts. In which, the number of sensitive hosts is 2 for scenarios with

less than 20 hosts, and 10% of normal hosts for scenarios with 20 or more hosts. In these scenarios, the number of subnets is set to 7 while the number of exploitable services running on each host is 2.

In each training session, there are 1000 episodes running. In each episode, the maximum number of actions the agent can perform is  $2 \times (S + 2H)$ . Hypothetically, the optimal number of actions the agent can perform is  $S + 2H$  which includes scanning all subnets at least once, scanning all hosts at least once, and successfully attacked the service on each host with just one attempt. Because this optimal value is difficult to achieve, the algorithm uses 2 times this value per episode.

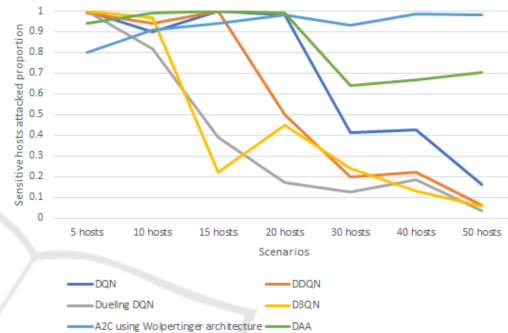


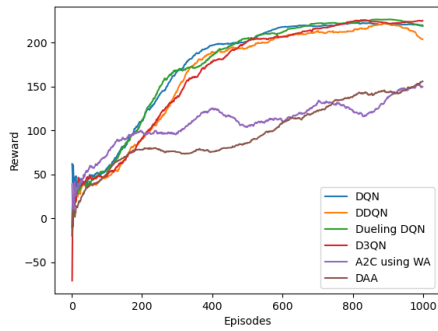
Figure 3: The sensitive machine attack proportion result of algorithms using different scenarios.

The figure 3 shows the results of algorithms with different scenarios. With the number of hosts less than 20, the results of the algorithms are not much different when DQN (Mnih et al., 2013), A2C using WA (Dulac-Arnold et al., 2015) and DAA all reach over 90%. However, with scenarios where the number of hosts is greater than 20, we find that the performance of A2C using WA and DAA far outweighs the DQN algorithms. In a scenario with 50 hosts, while the DQN algorithms have a sensitive host attacked proportion is less than 20%, the proportion in the DAA is 70% and A2C using WA remains at 99%.

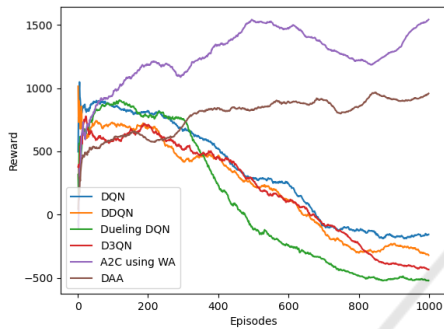
The reason the DAA in this experiment is not performing so well is that it used only 1000 episodes for training. Because of using both agents to learn the network, DAA is converging slower than DQN algorithm and A2C using WA. Therefore, the experiment using a small number of episodes has not shown all the capabilities of DAA.

The experiment result can be explained when we look at the reward received in the training of algorithms (figure 4). In a scenario with 5 hosts, we see that with 1000 episodes, DQN algorithms tend to learn and converge faster than A2C using WA and DAA. However, when the number of hosts is too large (50 hosts), the DQN algorithms are no longer able to





The training process of algorithms on scenario having 5 hosts



The training process of algorithms on scenario having 50 hosts

Figure 4: Rewards obtained by algorithms during the training process.

learn while A2C using WA and DAA still retain their ability to learn in the environment.

### 5.2 The Experiment Conducted on Large Network Environments

In this test, we still designed and implemented several networks on different sizes to test performance for A2C using WA and DAA. The other algorithms retain poor performance (0%) for scenarios having more than 50 machines, so we do not consider them in this experiment. Because we are testing on large network environments, the number of hosts (H) on each network this time is ranging from 64 to 1024 hosts. In which, the number of sensitive hosts is 10% of normal hosts for all scenarios. In these scenarios, the number of subnets is set to 20 while the number of exploitable services running on each host is 5. In each training session, there are 10000 episodes running. In each episode, the maximum number of actions the agent can perform remains  $2 \times (S + 2H)$ .

The figure 5 shows the results of A2C using WA and DAA with different scenarios. With a scenario with less than or equal to 128 hosts, both architectures have performance over 80%. However, with a scenario with a host of more than 512, A2C with WA

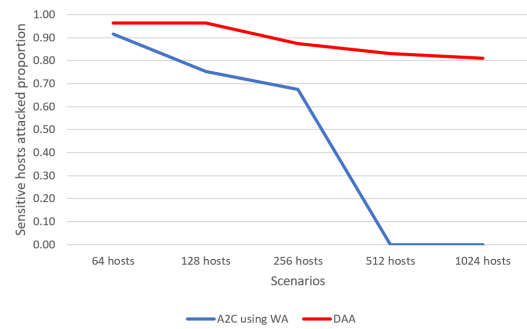
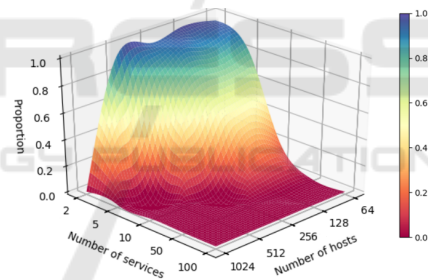


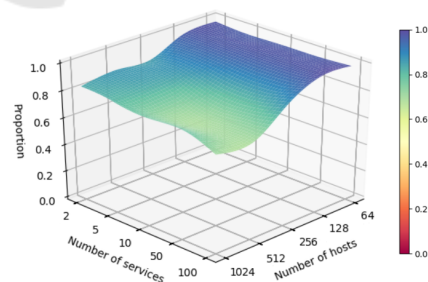
Figure 5: The sensitive machine attack proportion result of A2C algorithm using WA and DAA.

reaches its limit and is not capable of attacking sensitive hosts (0%). Meanwhile, the DAA kept its performance and the proportion of sensitive hosts being successfully attacked was still 81% in a scenario with 1024 hosts.

To better understand the scale-up of architectures, we keep the same parameters of the above experiment but change the number of exploit services running on each machine. The number of exploitable services (E) on each host this time is ranging from 2 to 100 services.



A2C using Wolpertinger architecture



A2C using DAA

Figure 6: Results of A2C algorithm using WA and DAA with different scenarios and number of exploitable services.

Based on the results showed in figure 6, we can see that the performance of A2C using WA is greatly affected by both the number of hosts and the number of exploitable services. As these two numbers increase,

A2C with WA is incapable of solving the problem. In contrast, DAA is slightly affected by the number of hosts but is not affected by the number of exploitable services. When the number of services running per machine increases, the DAA retains its performance around 70%, while WA fails to resolve these scenarios. Given these points, the data show that DAA outperforms WA in large environments.

The reason for such results is because the DAA uses two different agents to learn about the network. When the number of services increases in the range from 2 to 100, this complexity is still in processing capability of the exploiting agent, so the performance of DAA is almost not affected. Likewise, the structuring agent is capable of handling a number of hosts between 5 and 1024. In contrast, the agent of WA has to deal with host and service proliferation at the same time. Because the complexity increases exponentially when both of these values increase, the performance of WA drops rapidly to the point where it is no longer capable of solving problems. Therefore, DAA is superior to WA as well as other algorithms when performing PT in complex environments.

## 6 CONCLUSIONS

The study is focusing on the investigation into the application of RL to pentesting. The proposed architecture named double agent architecture is built based on two separate agents in order to improve the performance and accuracy of RL when applied to large network environments. The paper also conduct experiments to test the efficiency of DQN algorithms to evaluate its use in PT problems.

The main contribution of the paper is to increase RL's ability to solve the PT problem when the network is large. By dividing the PT problem into different subproblems including learning the structure of the network and learning how to choose the appropriate attack on the individual host, the double agent architecture has been proven to be impressive efficient. This method opens another approach when using RL to solve PT problems in the future.

Experimental results show that DAA outperforms other algorithms when solving PT problems with large networks. The size of scenarios can reach up to 1024 hosts and 100 services, and the DAA's ability to successfully attack sensitive hosts remains above 70%. With the number of exploitable services is less than 10, the performance of this architect with a network having 1024 hosts is up to 81%.

One of limitations of our work is that using the network simulator which is a high level of abstrac-

tion will cause a gap between studying and applying the problem in practice. The main direction for future work is proposed to use a more realistic environment such as VMs or real network as the input to the DAA. Frameworks and tools such as Metasploit and Nessus can be implemented to be able to more accurately evaluate method results in practice.

## REFERENCES

- Chui, M., Manyika, J., and Miremadi, M. (2016). Where machines could replace humans—and where they can't (yet). *McKinsey Quarterly*, 30(2):1–9.
- Dulac-Arnold, G., Evans, R., van Hasselt, H., Sunehag, P., Lillicrap, T., Hunt, J., Mann, T., Weber, T., Degris, T., and Coppin, B. (2015). Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*.
- Ghanem, M. C. and Chen, T. M. (2020). Reinforcement learning for efficient network penetration testing. *Information*, 11(1):6.
- Hasselt, H. V. (2010). Double q-learning. In *Advances in neural information processing systems*, pages 2613–2621.
- Hoang, V. N., Hai, N. N., and Tetsutaro, U. (in press). Multiple level action embedding for penetration testing. In *Proceedings of the International Conference on Future Networks and Distributed Systems*.
- Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285.
- Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937.
- Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.
- Phillips, C. and Swiler, L. P. (1998). A graph-based system for network-vulnerability analysis. In *Proceedings of the 1998 workshop on New security paradigms*, pages 71–79.
- Sarraute, C., Buffet, O., and Hoffmann, J. (2013). Pomdps make better hackers: Accounting for uncertainty in penetration testing. *arXiv preprint arXiv:1307.8182*.
- Schwartz, J. and Kurniawati, H. (2019). Autonomous penetration testing using reinforcement learning. *arXiv preprint arXiv:1905.05965*.
- Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., and Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003.