**Reinforcement Learning approaches for penetration testing.**

Table of Contents

[Declaration i](#_Toc95842141)

[Dedication ii](#_Toc95842142)

[Acknowledgements iii](#_Toc95842143)

[Table of Contents 1](#_Toc95842144)

[List of Figures 3](#_Toc95842145)

[List of Tables 4](#_Toc95842146)

[List of Abbreviations 5](#_Toc95842147)

[List of Symbols 6](#_Toc95842148)

[Title of Paper 7](#_Toc95842149)

[I. Introduction 7](#_Toc95842150)

[Background 7](#_Toc95842151)

[Aim of Research 7](#_Toc95842152)

[Structure of Paper 7](#_Toc95842153)

[II. Approach 7](#_Toc95842154)

[III. Experiments / Testing / Measurements / Data 7](#_Toc95842155)

[IV. Analysis and Discussion of Results 7](#_Toc95842156)

[V. Conclusions 7](#_Toc95842157)

[VI. Future Work 7](#_Toc95842158)

[VII. Instructions 7](#_Toc95842159)

[Formatting 7](#_Toc95842160)

[Title 7](#_Toc95842161)

[Headings 7](#_Toc95842162)

[Detailed instructions 7](#_Toc95842163)

[Indicative Length 7](#_Toc95842164)

[Use of colour 7](#_Toc95842165)

[Figures and Tables 8](#_Toc95842166)

[References 8](#_Toc95842167)

[Remove additional information 8](#_Toc95842168)

[Appendix A: Literature Review 9](#_Toc95842169)

[Introduction of Literature Review 9](#_Toc95842170)

[Body of Literature Review 9](#_Toc95842171)

[Conclusions from the Literature Review 9](#_Toc95842172)

[References of the Literature Review 9](#_Toc95842173)

[Indicative Length of Appendix A: Literature Review 10](#_Toc95842174)

[Appendix B: Data and Programs Used for this Report 11](#_Toc95842175)

[Required 11](#_Toc95842176)

[Optional 11](#_Toc95842177)

[Appendix C: Figures 12](#_Toc95842178)

[Appendix D: Tables 13](#_Toc95842179)

[References 14](#_Toc95842180)

List of Figures

*(required, include page numbers)[[1]](#footnote-1)*

Figure 1, Title of Figure #

List of Tables

*(required if you use tables, include page numbers)[[2]](#footnote-2)*

Table 1, Title of Table #

List of Abbreviations

*(required if you use abbreviations)[[3]](#footnote-3)*

List of Symbols

*(required if you use symbols)[[4]](#footnote-4)*

Selection of Reward in Reinforcement Learning for Auto penetration Testing

*Abstract*—Penetration testing has a main role to identify the security of network security and becomes more and more important as the interconnections by networks are increasing 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 rapid growth in all parts of science and technology, auto penetration testing methods based on AI have been developing. Especially, many researchers have interesting in reinforcement learning based auto penetration testing. There are many papers and packages for auto penetration testing with Reinforcement learning. However most of them concentrate on implementation of reinforcement learning model for penetration testing. To get optimal performance by reinforcement learning in penetration testing, it is very important to set the reward function correctly. In this paper, we develop the deep reinforcement learning based penetration testing method using efficient reward function.

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

# Introduction

The introduction to your paper should go here. This section usually has the following subheadings.

## Background

-Understanding for Penetration

{1} Computer networks are more than ever exposed to cyber threats of increasing frequency, complexity and sophistication [1]. Penetration Testing (shortly known as PT) is a well-established proactive method to evaluate the security of digital assets, varying from a single computer to websites and networks, by actively searching for and exploiting the existing vulnerabilities. The practice is an emulation of the operational mode that hackers follow in real-world cyber-attacks. In the current constantly evolving digital environment, PT is becoming a crucial and often mandatory component of cyber security auditing particularly after the introduction of the European GDPR (General Data Protection Regulation) for organizations and businesses. In addition to legal requirements, PT is considered by the cyber security community as the most effective method to assess the strength of security defenses against skilled adversaries as well as the adherence to security policies [2]. In practical terms, PT is a multi-stage process that often requires a high degree of competence and expertise due to the complexity of digital assets such as medium and large networks. {2} Penetration testing is different from security functional testing. The latter demonstrates the correct behavior of the system’s security controls while penetration testing determines the difficulty for someone to penetrate an organization’s security controls against unauthorized access to its information and information systems. It is done by simulating an unauthorized user attacking the system using either automated tools or manual method or a combination of both. The main goal of vulnerability assessment is to identify security vulnerabilities under controlled circumstances so they can be eliminated before unauthorized users exploit them. Computing system professionals use penetration testing to address problems inherent in vulnerability assessment, focusing on high-severity vulnerabilities. Penetration testing is a valued assurance assessment tool that benefits both business and its operations. From a business perspective, penetration testing helps safeguard the organization against failure through preventing financial loss; proving due diligence and compliance to industry regulators, customers and shareholders; preserving corporate image; and rationalize information security investment [4]. Penetration testing evaluates the effectiveness of existing security products and provides the supporting arguments for future investment or upgrade of security technologies. It provides a “proof of issue” and a solid case for proposal of investment to senior management [5]. {3}Auto

Presently, a number of tools have been developed that facilitate penetration testers and help improve their efficiency. These tools include network and vulnerability scanners as well as libraries of known security vulnerabilities ​[8]​. One of the most popular tools today is the open source Metasploit framework which has been in development since 2003 ​[8]​. The Metasploit framework contains a rich library of known exploits of system vulnerabilities along with other useful tools such as scanners, which are used for information gathering on a target. Tools such as these allow the pentester to work at a higher level of abstraction where they are mainly focussed on finding vulnerabilities and selecting exploits rather than having to work at the low level of manually developing exploits. This enables pentesters to work faster and also makes security assessment more accessible to non-experts ​[9]​. These tools have certainly been a boon to the cyber security industry, however, even with the great benefits these tools have provided they still rely on trained user, which are in short supply. Additionally, as systems grow in complexity the task of manually assessing security will become much harder to do systematically. One approach to trying to solve the problem of conducting efficient and reliable pen testing is to apply techniques from the Artificial Intelligence (AI) planning domain to pen testing in order to automate the process. The original concept for this took the form of “attack graphs”, which modeled an existing computer network as a graph of connected computers, where attacks can then be simulated on the network using known vulnerabilities and exploits ​[10]​. Attack graphs can be effective in learning the possible ways an attacker can breach a system, however using these graphs requires complete knowledge of the system, which is unrealistic from a real world attackers point of view, and also require the manual construction of the attack graph for each system being assessed. Another approach taken involved modeling an attack on a computer as a Partially Observable Markov Decision Process (POMDP) ​[11], [12]​. Modeling an attack as a POMDP introduces the attackers incomplete knowledge into the simulation and allows simulations to remove the assumption that the configuration of each host is known and instead models the observation of the configurations as the attack progresses. This approach can work well in practice against a single host machine but due to the computational properties of POMDPs, does not scale well ​[12]​. In order to produce a method for automating pen testing that can handle the uncertainty inherent in any system while still being computationally feasible more research into novel methods is required. A proposed solution to this problem is to simulate pen testing using an MDP. This approach would ignore the uncertainty about the state of the host computer's configuration and instead introduce the uncertainty into the success probability of each possible attack [13]​. This type of model is computationally more feasible than the POMDP approach and does not require complete knowledge of the network and host configurations.

However, it requires prior knowledge about the success probabilities of each possible action and it treats each target computer as exactly the same instead of utilizing information gathered about the target to produce more tailored attacks. Consequently, this approach addresses the issues of computational complexity and incomplete knowledge of network and host configurations but at the cost of accuracy of picking the best actions for each host. Another approach for solving problems that can be framed as a MDP that does not require information about the transition model of the environment is Reinforcement Learning (RL) ​[14]​. RL requires only the state space representation, the set of actions that can be performed and a reward function which defines what the RL agent is trying to achieve. The agent then learns a policy of actions to take from any given state through interaction with its environment. The use of RL has gained a lot of attention in recent years with its use in producing World Go champion beating agents ​[15]​, and although not as widely applied as other supervised machine learning approaches it has been successfully applied in the real world for in a number of robotics tasks ​[16]​. RL provides a more general approach when a detailed model of the environment is not known. In terms of automating penetration testing, we have information on the set actions that can be performed, and can frame the task into a reward function. However, due to the complex and ever changing nature of the cyber network environment, with constant updates to software and available exploits it becomes very difficult to maintain an accurate up-to-date model for the outcomes of performing any action. This combination sets RL as a good candidate approach for automated pen testing. However, RL offers its own challenges. It’s generality comes at the cost of requiring a large amount of data in order to learn the best policy of actions. This data is typically gained from using simulations in order to train the RL agent.

Recently, RL based penetration methods gets more and more attracts.

Numbers of packages for auto penetration testing using deep reinforcement learning are developed.

//Mulval, DEEP EXPLOIT,NAS

NAS is a python package which is given in the format of openAI gym Environment.

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, 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 for 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.

[] and [] developed 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 numbers of hosts are increasing.

The number of actions and states are given in the table 1.

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

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 RL algorithm to penetration testing on big networks, we should find the method to reduce the time that training to be 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 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 new method to solve the problems.

To do it, updating for Reward will be concentrated.

How to set Rewards for actions are important element 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 new deep RL based auto penetration testing method and compare the performance s.

## Structure of Paper

This paper constitutes 5 chapters.

Section 1 shows introduction to the paper.

Section 2 shows the approach for this paper.

Subsection 2.1 shows the

# Approach

Describe the approach you have taken to do your research

+

As described in introduction, there exists 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 environment for pen-testing. NAS generates gym Environment so we can easily use deep learning python packages such as tensorflow, pytorch and openAI which are widely used to.

This section has two sub sections.

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

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)​.

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

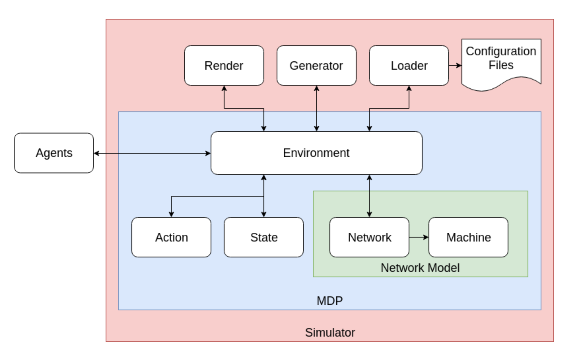


Figure . Structure of penetration testing based on DQN

* 1. **Environment**

1. ***Network Model***

The network model defines the organization, connection and configuration of machines on the network and is defined by the tuple ​{subnetworks, topology, machines, services, firewalls} ​ . An example network made up of five subnetworks, 11 machines and firewalls between each subnetwork is shown in figure 3.2.1. This network could run any number of services as this is defined at the level of machine. We provide more details for each component in the following paragraphs. This model aims to abstract away some details of a real-world network that are not required when developing autonomous agents such as specific types of connections between machines and the location of switches and routers in the network. The reason for this abstraction is to try and keep the simulator as simple as possible and at the level of abstraction that the agent is expected to work at which is determining which scans or exploits to use against which machine and in what order. The specific details of performing each action, for example which port to communicate with, are details that can be handled by application specific implementations when moving towards higher fidelity systems. Penetration testing is already moving in this direction with frameworks such as Metasploit which abstract away exactly how an exploit is performed and simply provide a way to find if the exploit is applicable to the scenario and launch it, taking care of all the lower level details of the exploit ​[18]​. This simpler network model is also used in order to keep it as general and easily scalable as possible.

* Sub-Networks

Each network is made up of multiple sub-networks or subnets. A subnet is a smaller network within the larger network that is composed of a group of one or more machines that are all able to communicate fully with each other. Each subnet has its own subnet address, which is indicated as the first number in any machines address (e.g. the 4 in the address (4, 0)). This is a simplification of IP addresses which use a 32-bit string and a seperate 32-bit subnet mask to define the network, subnet and machine address. For the purpose of the NAS, it makes sense to use a simpler system since we are only dealing with a single network as opposed to IP addresses which deal with millions of machines on thousands of networks across the internet. Although all machines within a subnet can communicate fully, communication between machines on different subnets is restricted. Inter-subnet communication is controlled by the network topology and firewall settings.

* Topology

The network topology defines how the different subnets are connected and controls which subnets can communicate directly with each other and with the external network. As an example, in the network in figure 3.2.1 subnet 1 is the only network that is connected to the external world and subnets 1, 2 and 3 are all connected to each other while its only possible to communicate with machines on subnets 4 and 5 via a machine on subnet 3. In this way an attacker may have to navigate through machines on different subnets in order to be able reach the goal machines. We can view the network topology as an undirected graph with subnets as its vertices and connections as edges.

* Machines

Machines The most primitive building block of the network model is the machine. A machine in the NAS represents any device that may be connected to the network and hence be communicated with and exploited. Each machine is defined by its address, in the form of a (subnet\_ID, machine\_ID) tuple, it value and it’s configuration. An example machine definition can be seen in figure 3.2.3. The value of a machine is defined by the user with higher values given to sensitive machines, that is machines that the attacker wants to gain access to or that the owner wants to protect. Each machine runs services that can be communicated with from other machines within the same subnet or on neighbouring subnets, firewall permitting. The services available on each machine define its configuration and each machine on the network will not necessarily have the same configuration. This is included since not every machine on the network will be the same as some can be expected to be used for different purposes E.g. Web servers, file storage, user machines. The services present on a machine also define its points of vulnerability, since the services are what the attacker is aiming to exploit.

* Services

Services are used to represent any software running on a machine that communicates with the network. They are analogous to software that would be listening on an open port on a computer or connected device. Within the NAS services are considered to be the vulnerable points on any given machine, and can be thought of as services that have a known exploit which the attacker is aware of. In a real world scenario it would be the same as keeping track only of services that an attacker has a known exploit for, while ignoring any other non-vulnerable services. Based on this reasoning within the NAS, we assume each service is exploitable by one action, so the agents job is to find which service is running on a machine and select the correct exploit against it. Each service is defined by a unique ID, the probability its exploit will succeed and also the cost of using the exploit. Figure 3.2.3 shows an example machine in a network scenario where the attacker has exploits for the ftp, ssh and http services, while figure 3.2.4 shows the set of exploitable services and the associated success probability and cost of their exploits. The ID of each service can be any unique value and does not necessarily have to be a name related to a real world service. In this way it is easy for the NAS to generate test scenarios with any number of machines and services to aid in testing the scaling performance of agents by simply generating service IDs as needed. When investigating the application to more real world settings, the ID would be replaced with a specific service name and version, so it would be possible to track vulnerabilities and know what services require patching (e.g. Samba version 3.5.0).

* Firewall

The final component of the network model are the firewalls that exist along the connections between any subnets and also between the network and the external environment. Firewalls act to control which services can be communicated with on machines in a given subnet from any other connection point outside of the subnet. They function to allow certain services to be used and accessed from machines within a subnet with the correct permissions, while blocking access to that service from unwanted entry points. Each firewall is defined by a set of rules which dictate which service traffic is permitted for each direction along a connection between any two subnets or from the external network. Figure 3.2.5 shows an example firewall that sits between subnets 1 and 3 and which allows access to the ssh service on machines on subnet 3 from machines on subnet 1 and access to ftp and http services on machines on subnet 1 from machines on subnet 3. In a real world setting firewall rules are typically set by defining which port can be accessed, however for simplicity and since for most cases the same services are run on the same port numbers, we have decided to instead define rules by service rather than port.

* Scenarios

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.

Following figures show the structures of networks used for testing.

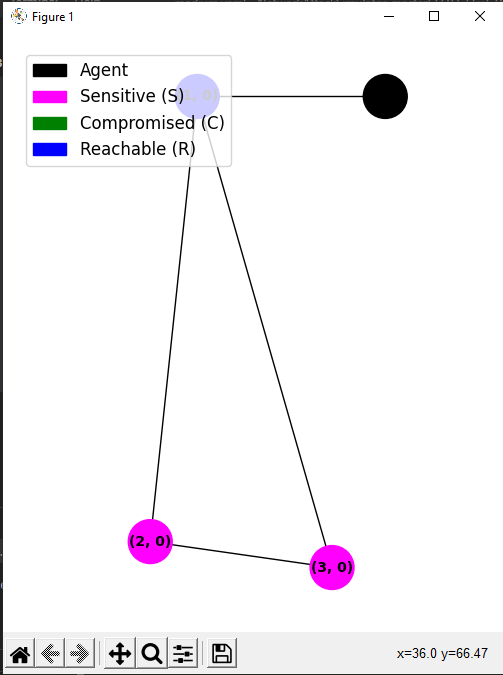


Figure . Tiny Scenario

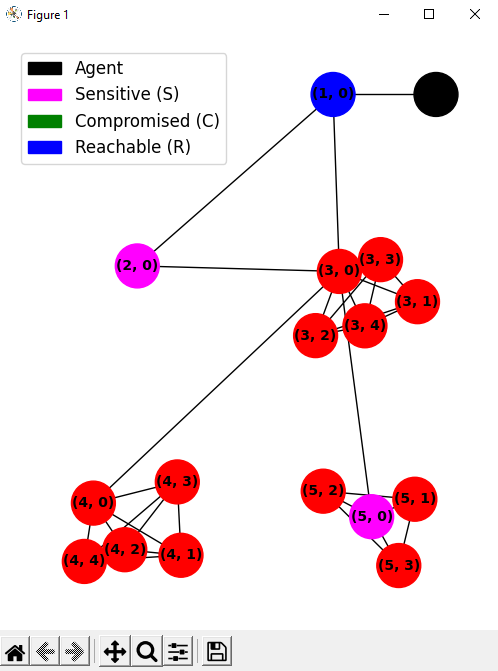


Figure . Scenario 2

Tiny scenario contains three subnetworks and three hosts.

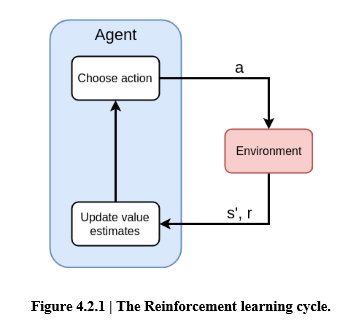
Medium scenario contains 5 subnetworks and 16 hosts.

The numbers of sensitive hosts are 2 for both scenarios.

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

1. ***Environments for DQN***

RL algorithms learn optimal policies through interaction with the environment. This done by starting from some initial, typically random, policy then iteratively learning the values of taking a certain action for a given state, ​Q(s, a), ​ by choosing an action based on the current policy, applying that action to the environment, then updating the state-action value, ​Q(s, a) ​ , based on the received experience (fig. 4.2.1) ​[14]​. Specific RL algorithms differ based on how they choose actions, update their value estimates for the value function and the form of the value function. We make use of Q-learning in our RL implementations for the value update strategy. Q-learning is an off-policy temporal-difference algorithm for learning the action-state values and is defined by the update function in equation (4.3) ​[44]​. ​ Where α is the step size, which controls how much to move current estimate towards the new estimate and 𝛾 is the discount factor which controls how much to weight immediate versus future rewards. Q-learning has been shown to converge to the optimal state-action values, as the number of visits to state-action pair approaches ∞



* NAS: States

A state, ​s ​ ∊ ​ 𝓢​ , is defined as the collection of all known information for each machine on the network. That is, the state includes for each machine, if the machine is compromised or not, reachable or not and for each service, whether the service is present, absent or its existence is unknown. A machine is considered to be compromised if an exploit has successfully been used against it. While a machine is considered to be reachable if it is in a subnet that is publicly accessible (connected directly to external network), in the same subnet as a compromised machine or in a subnet directly connected to a subnet that contains a compromised machine. The state space is therefore all possible combinations of compromised, reachable and service knowledge for each service and for each machine. Hence, the state space grows exponentially with the number of machines and services on the network. Equation (3.1), shows the size property of the state space, ​| 𝓢 | ​ , where ​|E| ​ is the number of exploitable services and ​|M| ​ is the number of machines in the network. The base of for the exponential is 3, since for each exploitable service the agents knowledge can have one of three values: ​present ​ , ​absent ​ or unknown.

An example network and an associated state are shown in figure 3.2.6. In this state the attacker has successfully compromised the machine at address ​(1, 1) ​ and so can now communicate with machines on subnets 2 and 3, which is indicated by reachable being set to ​true for each machine on those subnets. Additionally, the configuration of the machine at ​(3, 1) ​ is known which would have been gained through a scan action, while the configuration for machine ​(2, 1) ​ is ​unknown ​ . Note that the state does not include any information about the firewall settings, since this would require privileged access to determine. For this simulator we assume it is not possible for the attacker to gain this information and it must instead learn it indirectly through the success and failure of exploit actions.

* NAS: Actions

The action space, ​ , ​ is the set of available actions within the NAS and includes a single scan action and an exploit for each service and each machine on the network. The scan action is designed to mimic the Nmap complete scan, which returns information about which services are running on each port of a given machine and also versions of each service ​[19]​. In reality more targeted scanning may be required to discover complete information about specific services, but for many use cases, Nmap scans return the information required to determine which service is running. Scan actions are considered to be deterministic, always returning information about the presence or absence of a service. For each possible service on the network, there is a matching exploit action. Each exploit action can be deterministic or non-deterministic depending on the configuration of the environment chosen by the user. A successful exploit action will result in the target machine becoming compromised. The success of any exploit is determined by whether the target machine is reachable, the target service is present, if that service is blocked by the firewall or not and also the success probability of the action. Each action also has an associated cost which can be set when configuring the environment. This cost can be used to represent any metric such as the time, skill, monetary cost or noise generated for a given action, depending on what performance metric is trying to be optimized for.

* NAS: Rewards

The reward function is used to define the goals of the autonomous agent and what is trying to be optimized by the agent. The reward is defined over a transition ​ 𝓡 (s, a, s’) ​ , so starting from one state ​s ​ taking action ​a ​ and ending in the resulting state ​s’ ​ (eq. 2. ). The reward for any transition is equal to the value of any newly compromised machine in the next state ​s’ ​ minus the cost of action ​a ​ . So if no machine was compromised, then the reward is simply the cost of the action performed. With this reward function, the goal of the attacker becomes to try and compromise all machines with positive value on the network while minimizing the number or cost of actions used. This mimics a real world nefarious attacker, whos goal we assume is to to retrieve privileged information or gain privileged access on the system. Using the NAS it is possible to set these goals by changing the value of certain machines on the network, so for example machines that may contain sensitive documents or contain privileged control on the network.



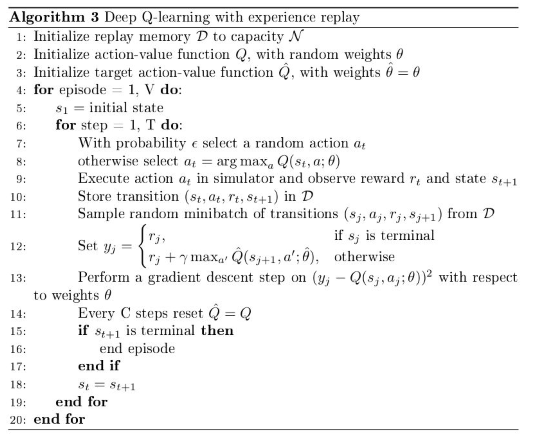
Where, value(s’, s) ​ returns the value of any newly compromised machines in ​s’ ​ from ​s ​ or 0 if no new machines were compromised and ​cost(a) ​ returns the cost of action ​a.



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

* Deep Q learning algorithms

DQN algorithm which is used for penetration testing is shown in figure .



* Steps for Testing.

Tests are devided 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

Main parameters for train are as following.

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

For tiny scenario, since there are 56 observations and 18 actions, there exist 56 inputs and 18 outputs for neural network. We set the numbers of hidden layers as 1 and the number of cells as 32.

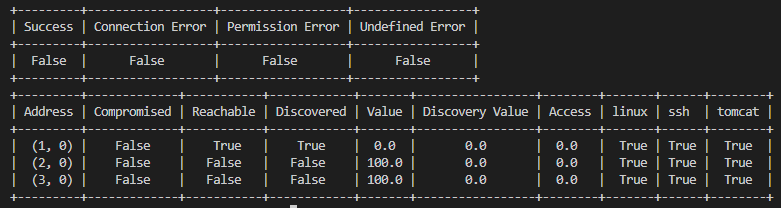
Training steps are set as 10000 and target update frequency is 1000.

Once training ends, we test the DQN agent.

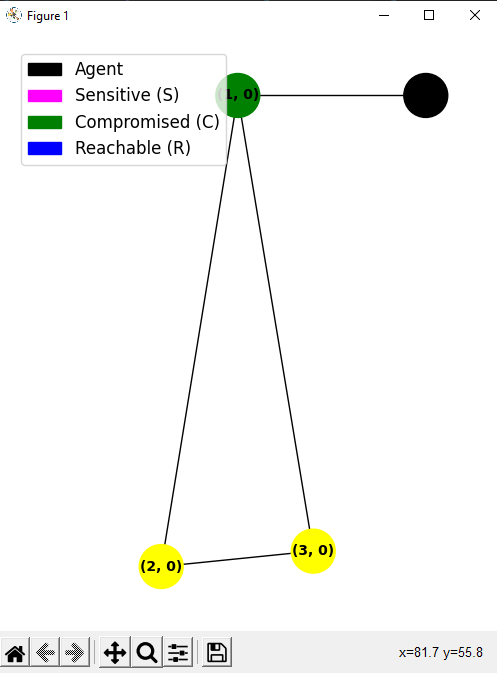
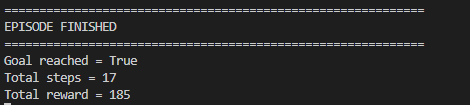
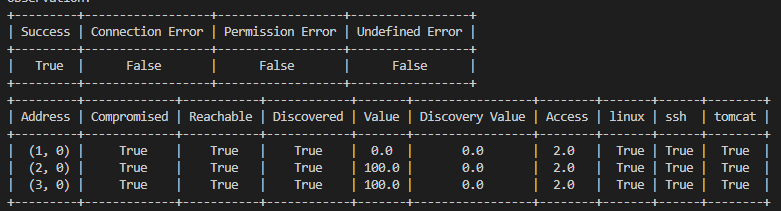
Initial states of networks are given as following figure.

* Rewards

We used the rewards given in the NAS environment .



Final states of networks are given as following.



As we can see from the result, DQN agent find the attack path correctly and reached a goal.

All sensitive nodes are compromized.

Next we test DQN agent for medium scenario.

The structure and initial states of networks in this scenario are as following.

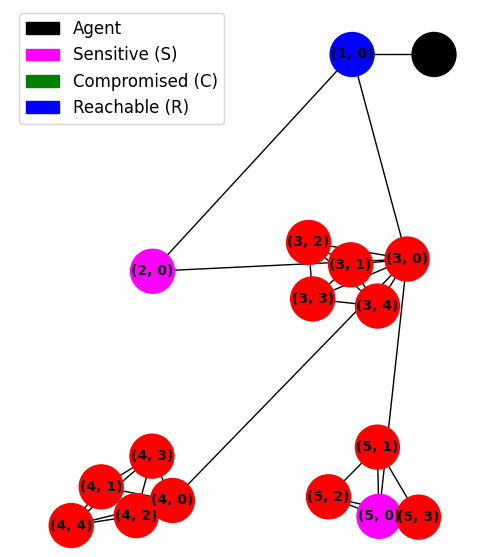
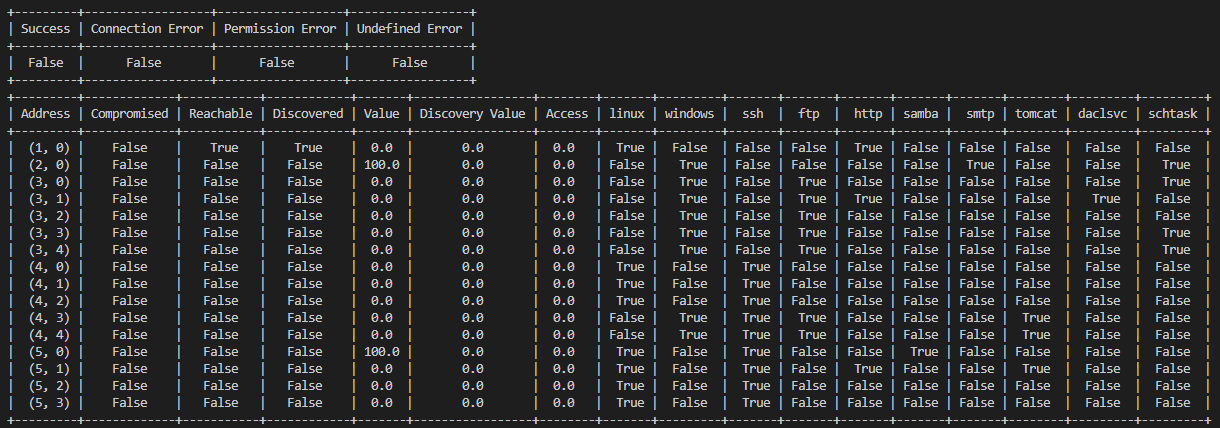


Fig .

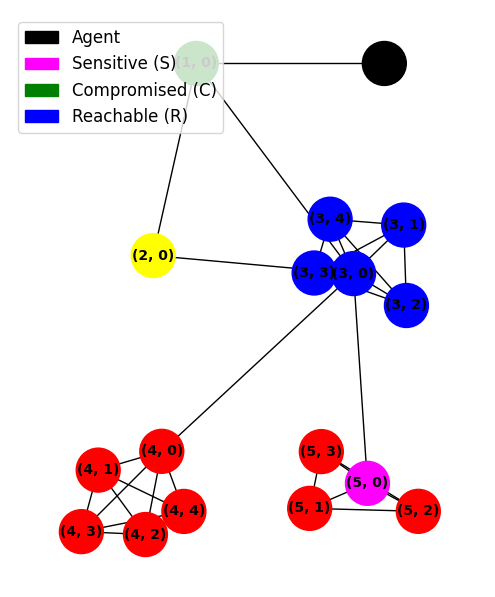
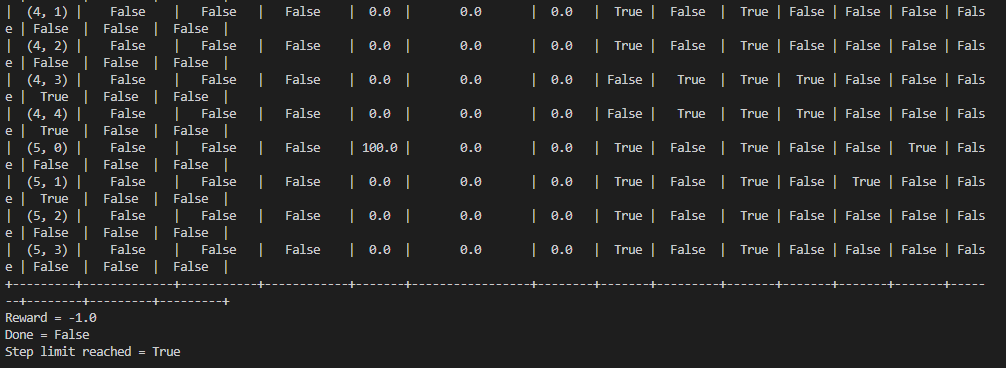
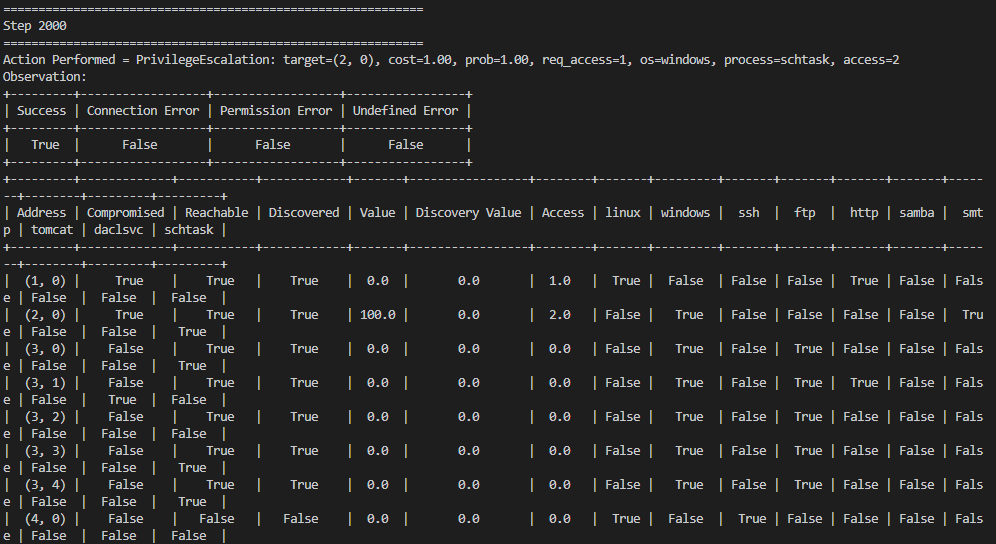
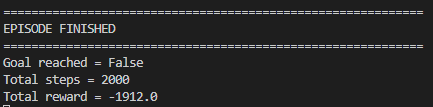
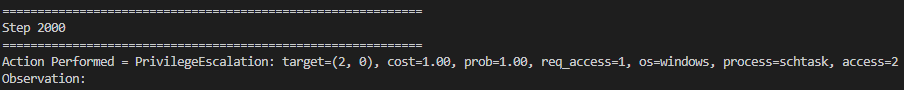
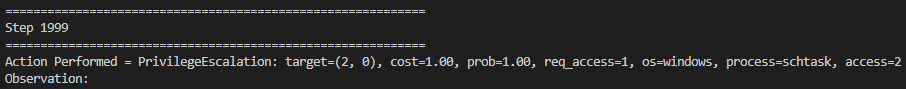
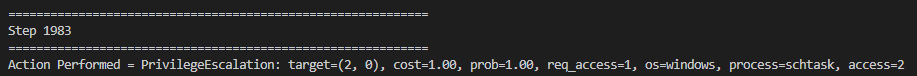


Fig

Since the numbers of actions are 192, we set the number of hidden layer as 1 and the numbers of cells as 256.

Training steps are set as 1000000.

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



As we can see from the figures, test didn’t reach goals.

From the transitions of states and actions, we can see that agent selected the same action escalation for host {3,0} for last steps.

It means that the agent is put in a infinite loop.

We performed serveral tests on this agent with same environment, however tests showed the similar results and agent didn’t reach the goals.

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

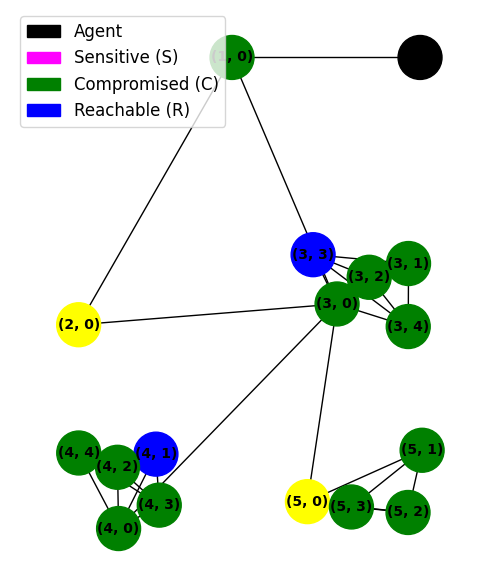
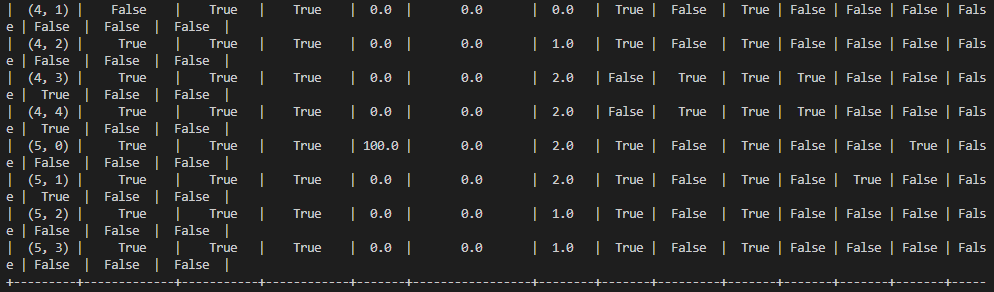
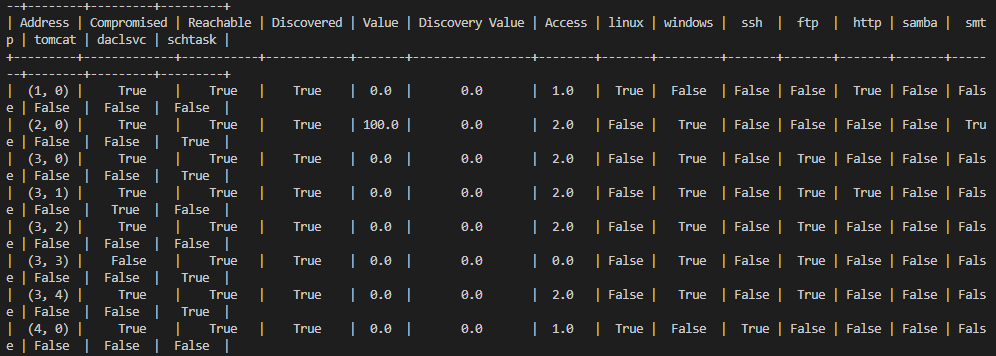
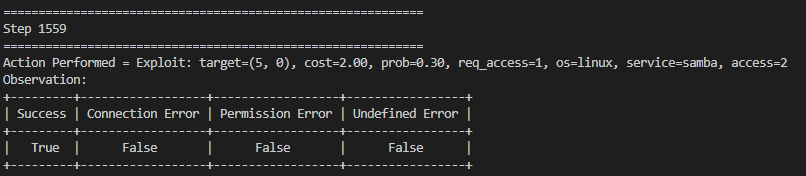
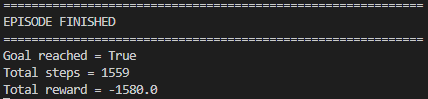
To prevent this, we can set the epsilon as a large value.

However it will produce another problem.

If epsilon is set as a large value, the performance of agent to avoid infinite loop will be upgraded however it will degrade the agent’s performance in totally since it will increase the probability to select random action.

We performed test to avoid infinite loop by selecting random agent if the same action is selected repeatedly.

Results for this case are shown in the following.



As we can see from the result, agent avoid from 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 ideal optimal path is 10 and 192 . We tested for this agent several times and get similar results.

|  |  |  |
| --- | --- | --- |
|  | Total Steps | Total Reward |
| present | 1371.45+/- 420.41 | -1875.29+/- 660.62 |
| Preceding one | 1401.34+/- 317.23 | -1902.32+/-603.47 |

This result is similar as preceding one in [].

Although a goal is reached, the length of path is too long than optimal path and reward is also too low.

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

In the next sub section, we are going to try to analysis the reasons of these problems and find the solution to solve it.

* 1. Analysis

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

As we described in above sections, the numbers of states and actions are increasingly rapidly by the numbers of hosts. In fact, net experts who have full knowledge for 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 DQN agent largely.

This can be implemented by change reward function of 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 environment for rewards.

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

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

Updating for the NAS environment is described in subsection.

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

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

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

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

Rewards for current environment are seen as following.

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

|  |  |
| --- | --- |
| Case | cost |
| Exploit(e\_ssh) | 3 |
| Exploit(e\_ftp) | 1 |
| Exploit(http) | 2 |
| Exploint(e\_samba) | 2 |
| Exploit(e\_smtp) | 3 |
| Privilege\_escalation | 1 |
| Service\_scan\_cost | 1 |
| OS\_scan\_cost | 1 |
| Subnet\_scan\_cost | 1 |
| Process\_scan\_cost | 1 |

From tables, we can see that reward can only represent the information host is compromised or not but other attacking method information. So DQN agent can’t decide correct action for given environment.

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

This is the reason that DQN agent worked badly for big networks. From analysis for 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 big network.

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

Network attacking has 6 steps of Exploit, privilege escalation, subnet scan, os scan, process scan, service scan on NAS.

First step of attacking is exploit with several protocols such as ssh, ftp, http, samba and so on.

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

Network experts firstly find the 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 expert will not try privilege escalation for not exploited hosts.

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

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

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

|  |  |
| --- | --- |
| 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 table, reward value is set reflecting network step.

Exploit is most important step and also first step.

Since success exploit gets greater reward value than others, DQN agent avoids to be put in infinite loop.

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

It is since attacks to sensitive hosts are our goal.

It is expected that once we use these rewards, training can get better performance than 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 attacking path.

We are going to solve this problem by applying cost value as in the preceding one.

However we don’t use the cost value as in the preceding one. Since we set the value based on network attacking methods, there is not mean to set the cost differently for attacking method.

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

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

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

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

To see the effects of proposed reward function, we firstly use only 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.

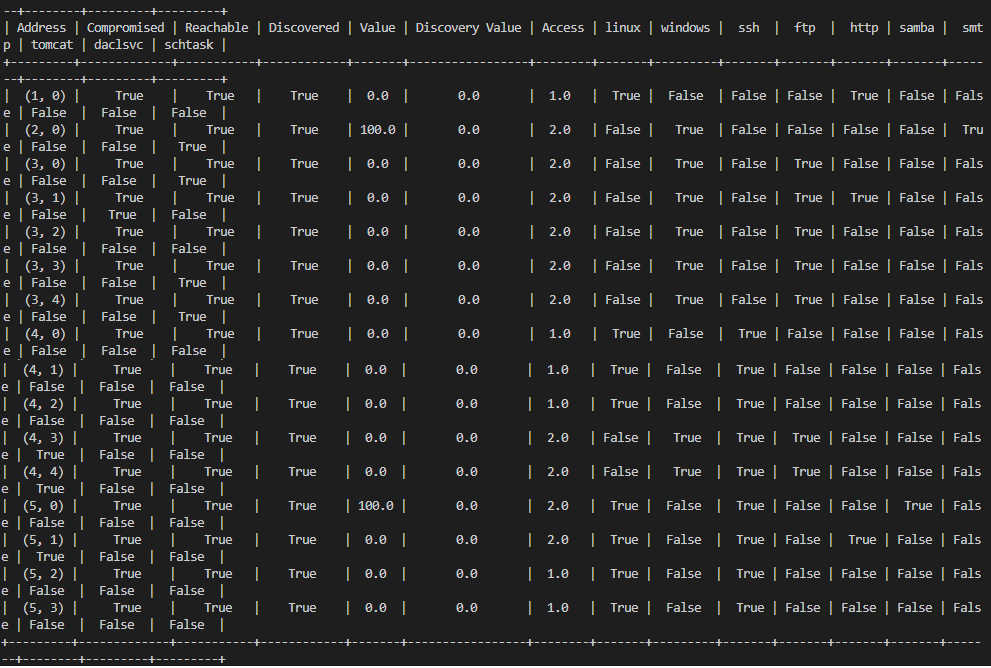
# Experiments / Testing / Measurements / Data

Describe how you have gathered the data for your research

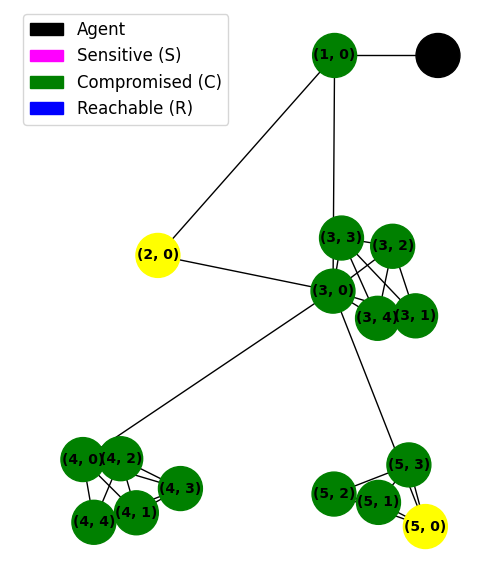
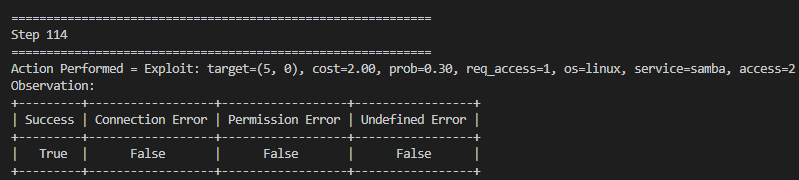
Firstly we perform experiment with the first term of reward. We expected that the training will be success to make DQN agent to reach a goal though it will not find optimal solution.

We change the reward value in the NAS environment and test with same scenario file.

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



I



As we can see from the result, steps are taken 114 and reward is 76. All nodes are compromised in this experiment. It shows that proposed algorithm gets perfect efficient to make DQN agent to 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.

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

|  |  |  |  |
| --- | --- | --- | --- |
| 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, goal is always reached and total steps are in range (37,273) for 10 times experiments.

Rewards are in range (-83 to 127).

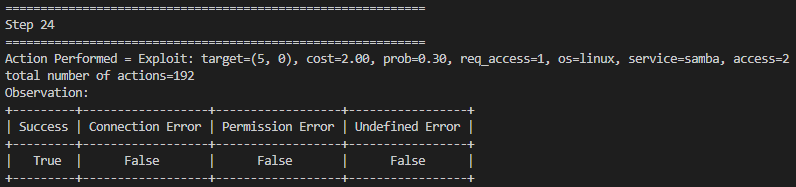
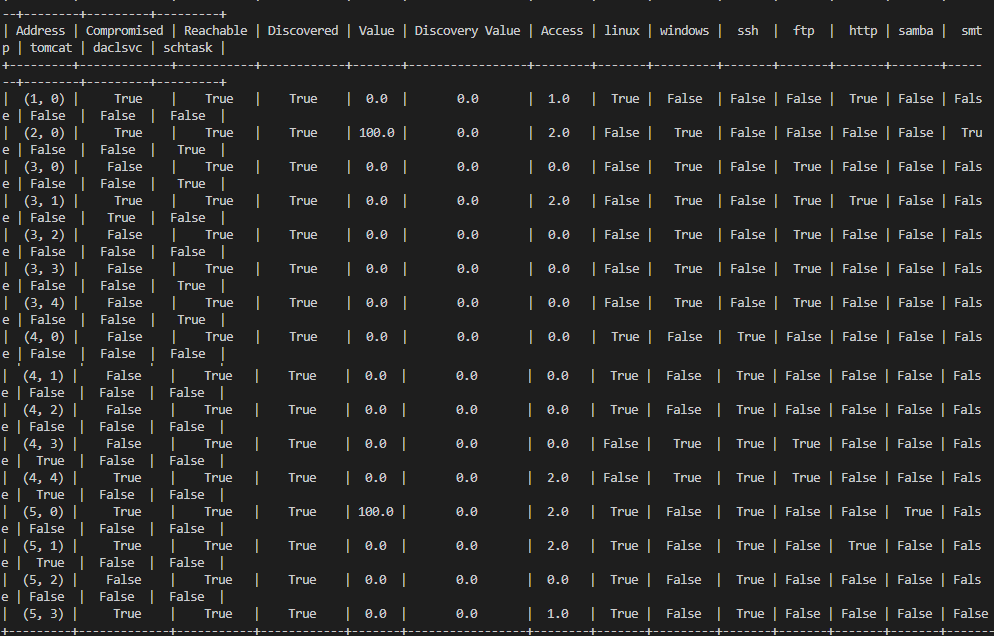
As we described in the previous tests for original algorithm, total steps are reduced 10 times at least and total rewards are increased largely.

Now, we can go into second step experiment.

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

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

We perform experiment as the same ways as in first experiment.



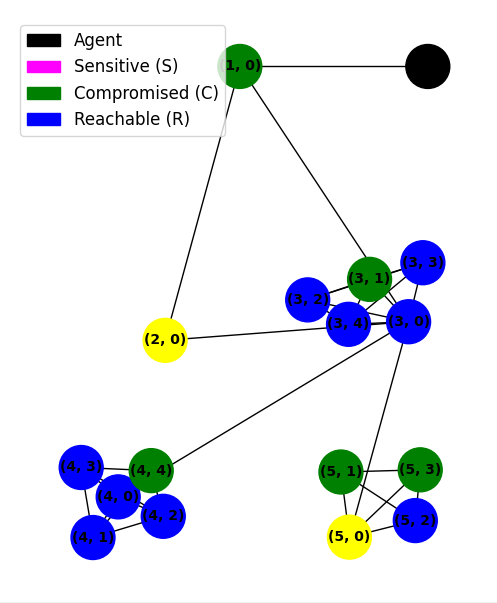
As we can see from the result, total steps are reduced to 24 and reward are increased to 172. This result can’t be get in the previous experiments with maximum value.

Compared to ideal optimal reward 192, it is a very high reaward value. Totol steps are only 24.

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

Test resuls are shown in the following table.

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |



# Analysis and Discussion of Results

We are going to analyse the performance of proposed algorithms with experiment results.

We are going to calculate the mean, variation for rewards and steps and compare them with original one.

Following table shows the performance for several methods.

|  |  |  |
| --- | --- | --- |
| Method | Total steps | Total rewards |
| Preceding one | 1371.45+/- 420.41 | -1875.29+/- 660.62 |
| Preceding two | 1401.34+/- 317.23 | -1902.32+/-603.47 |
| Proposed one | 143.500+-27.84 | 46.700+-27.72 |
| Proposed two | 63.400+-14.47 | 129.400+-15.21 |

Results of experiments show that prosed algorithm can upgrade the performance of DQN agent largely.

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

# Conclusions

In this paper we review for the researches for auto penetration testing based on DQN agent and find the drawbacks.

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

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

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

From experiments we validated that proposed DQN agent has ability to work well instead network experts for big networks.

# Future Work

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

We concentrate on update 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 for to reflect network attacking methods to other penetration testing tools which use RL methods.

Appendix A: Literature Review

{4}This work is rooted in a long line of applied research works on automating and optimizing offensive cyber-security auditing processes and systems especially vulnerability assessment (VA) and PT [2,11]. Among the most signiﬁcant contributions in this regard, we present here a summary of previous research with a special focus on the adopted approaches and the contributions. Initially, researchers were interested in the planning phase. Some works were implemented within the industrial PT systems and frameworks while others remained research ideas [2,4]. As PT automation and enhancement domain is situated between both cyber-security and AI research ﬁelds, several axes of research were addressed and progressed throughout different research ﬁelds and methodologies of automated planning (sub-area of AI) [6,7]. Early research focused on modeling penetration as attack graphs and decision trees reﬂecting the view of PT practice as sequential decision making [8]. Practically, most of the works were more relevant to vulnerabilities assessment than to PT [4]. Among themostsigniﬁcantcontributionsinthisregard,wepresentinthisliteraturereviewsectionasummary of the previously completed research with a special focus on the adopted approaches and the contributions. For the purpose of clarity, we start by addressing the full picture of the research in this ﬁeld and proceed later into dividing the research axes by type, methodology, and approach [6].

1. Previous Works on PT Automation Automation is an obvious approach to adopt for PT tasks when the objective is to produce highly efﬁcient PT systems. Nonetheless, automating the entire process of testing including the versatile tasks and sub-tasks for each phase is challenging and often fails to reach the objective if done in an inappropriate way, e.g., the use of automated tools and systems which blindly perform all the possible and available tests without any optimization or pre-processing [6,7]. The automated systems require the permanent control of a human PT expert and often fail to produce acceptable results in medium and large assets context because of the signiﬁcant number of operations required to cover the entire network [8–11]. In addition to the required time which surpasses the realistic duration of tests, more others issues are created by automation such as the generated trafﬁc (networkcongestion) and the high number of false positives alerts triggered on the asset defense solution such as IDPSs and ﬁrewalls. The use of PT blind automation was limited to small networks and some medium-size networks with the use of customized scripts which are inconvenient requiring substantial effort as well [2]. Early research focused on improving PT system by optimizing the planning phase which was modelled as attack graphs or decision trees problem which reﬂects the nature of PT practice as sequential decision making. Most of the works were nonetheless relevant to vulnerabilities assessment (VA) rather than PT because of the static nature of the proposed approach and its limitation to the planning phase [4]. Amongst the most signiﬁcant contributions, we ﬁnd the modeling of VA as attack graphs in form of atomic components (actions), pre-condition and post-condition to narrow the targeted vulnerability [8] but this approach was more an application of classical planning methods in order to ﬁnd the best attack graph. Further similar works were carried out on automating the planning of PT tasks but blind automation did not address the problem of enhancing performances and only covered the planning phase of PT practice [7,10]. Nevertheless, a remarkable work on optimization was introduced in [4] by modeling PT as planning domain deﬁnition language (PDDL) which for the ﬁrst time accounted for attacking and post-attacking phases of PT in addition to the ﬂexibility offered by the solution which enabled integration with some PT systems. The proposed solution generated different type of attack plans (AGs) for single and multi-paths PT scenarios which was then directly implemented within the attacking and exploiting system and executed along with interacting with information-gathering tools for transforming the information acquired during that phase into input to a planning problem to be solved separately and then used by the attacking system for the purpose of optimization. The only drawback of this approach was the scalability which was fatal as it was only limited to small and medium-size networks [6]. AI was also considered to improve PT practice in some research [6,7] but most of the proposed modeling approaches failed to deal with the persistent uncertainty in PT practice and especially the lack of accurate and complete knowledge about the assessed systems. An exception was the use of ML algorithmswithinaprofessionalPTandVAsystemcalledCore-ImpactinwhichthePTplanningphase was modeled as a partially observable Markov decision process (POMDP) solved using an external POMDP solver to determine the best testing plan in form for attack vectors. However, the proposed model itself is questionable as it obviously fails to model the full PT practice and thus can not cover the remaining testing phases and tasks especially the vulnerability assessment, testing and pivoting phases known to be highly interactive, sequential and non-standard compared with the planning and information-gathering phases [12].
2. Drawbacks and Limits of the Current PT Practice In this subsection, we will present an overview of the domain of PT and the automation of the practice along with highlighting the limitations of the current automated frameworks, systems, and tools. Penetration testing often involves routine and repetitive tasks which make it particularly slow on large networks. These tasks are unfortunately crucial for the practice and cannot just be dismissed although much of this routine can be automated [6]. Although the proposed solutions were in theory very relevant and seemed to solve the problem, the PT practice demonstrated that the brought improvements were not enough to solve the core issue in the practice which time and resources. Some solutions were, on theother hand, fundamentally unﬁt and inadequate for PT context. It is obvious that human capabilities and performance are limited when it comes to large and complex tasks compared with a machine especially with today’s computing power [11]. The average penetration tester can spend days or weeks in testing a medium-size LAN (we are concerned here with comprehensive testing when the entire network is covered). In addition to the time and effort allocated, a considerable amount of systems downtime will be accounted for as a result of the performed tasks [1]. The ﬁrst two points add to poor performance in term of results quality and accuracy including error and omission which could be crucial resulting from the fact that human makes mistakes, change opinion and get bored [13]. PT automation (automated systems and tools) were presented as a magic solution to the mentioned issues. A fully or even semi-automated solution was thus developed aiming to reduce human labor, save time, increase testing coverage and testing frequency, and allow broader tests. The proposed solution was very diverse in terms of adopted approach when some relied on automated planning (phase 1) by generating automatically attack plan (named attack graph) and then executing the attack in an automated manner. Other solutions were more creative and attempted to mimic the whole process and automate the system to carry out complex (chained) penetration testing tasks and use more exploits [14]. The cybersecurity research community starts questioning the limits of the existing PT systems, frameworks and tools which are expected to become more automated and perform most of PT tasks with little or without human intervention and especially during the ﬁrst two phases of PT: information gathering and vulnerability discovery. Organizations with a constant need for internal security auditing are, on the other hand, interested in more efﬁcient PT systems that are fully automated and optimized to perform basics and repetitive PT tasks without human intervention [11]. Nonetheless, researchers were struggling with automation as PT practice is a complicated process that humans barely master. Therefore designing a machine to replace PT experts is a challenge given the multi-phase nature of PT practice with high dependency between the different phases and tasks. Besides the complexity of PT practice, the information handled is another major issue as PT reconnaissance and information gathering phase usually produces an incomplete proﬁle of the assessed system and fail to yield a complete knowledge. This issue is often dealt with by an expert by repeating some tasks, changing approach, or simply making assumptions and continuing the tests. On top of the classic complexity associated with PT, modern attacks adopted evasive techniques andcomplexattackingpathsthatallowthemtoevadenetworkandsystemsdefenses. Skilled attackers would usually seek to achieve their goals through the exploitation of a series of vulnerabilities as part of a chain of sub-attack which enable them to can take advantage of hidden (non-obvious) and composite vulnerabilities (composed of a chain of harmless ﬂaws when together become an exploitable vulnerability) in networks. Each part of the infrastructure or systems may be approved to be secured when considered alone, but their combination and interaction can often provide a pathway for an opportunistic attacker. The ability to detect and analyze the interaction of multiple potential vulnerabilities on a system or network of systems leads to a better understanding of the overall vulnerability of the assessed system [7]. Finally, PT output data is a crucial issue because it is currently not used properly during retesting or future tasks and simply discarded after the PT report generation. In the cybersecurity context, only a few security conﬁgurations and system architectures change over the short and medium-term. Therefore most outputs of previous tests remain applicable whenare-testing is required[12]. This particular problem is one of the key motivations of our research.
3. Motivation and Contribution Generally, complexity is the worst enemy of control and therefore security; computer networks are not an exception to this rule. During the last decade, protecting and defending networks and critical digital assets from cyber threats required the security professional to consider a less classic approach (avoiding the trap of bolting on more and more security layers and policies). They turned their attention toward offensive security. Cybersecurity researchers were confronted with the need of an intelligent PT system and framework to support human experts in dealing with high-demand on PT and the associated complexity and risks by allowing systems to take over human and conduct some or all of the PT tasks notably reconnaissances, information gathering, vulnerabilities assessment and exploiting and therefore leave experts focusing on more complex issues such as post-exploiting and testing complex attacks [10,12]. Given the aforementioned facts about PT practice, ML seems to be the best answer to our problem. In fact, several AI techniques were initially considered but only machine learning through reinforcement learning was selected as the most promising option to allow an automated PT system to behave like a real tester in terms of operation and gaining skills gradually along with practice. This researchcomestobridgethegapinthecurrentPTpracticeandwillaimtoresolvethefollowingissues:

• Reducing the cost of systematic testing and regular re-testing due to human labor cost;

• Reduce the impact on the assessed network notably the security exposure, performances, and downtime during the testing; • Relieve human experts from boring repetitive tasks and assign them to more challenging tasks; • Dealing more effectively with changing cyber threats by allowing ﬂexibility and adaptability; • Perform more broad tests by covering a wide variety of attack vectors and also consider complex and evasive attacking paths that are hard to identify and investigate by human testers.

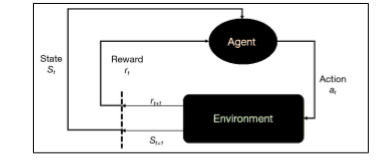
To sum up, cyber hackers seek to achieve speciﬁc goals through the exploitation of a series of vulnerabilities as part of a chain of sub-attacks. IAPTS is intended to do the same by autonomously discover, exploit and control networks machines and networking equipment along with revealing hidden (non-obvious) and complex vulnerabilities in network infrastructure or segment. In other words, IAPTS should cover the entire PT phases and not be limited to the vulnerability assessment proposed so far by [1]. It will also allow more testing coverage and exploration to deal with commonly locally-secured systems that are approved to be secured when considered alone, but the combination of the interaction can result in opening a pathway for an attacker. Finally, the PT expertise extraction, generalization and re-utilization is the ultimate contribution of IAPTS to the current PT practice [12].

The design and analysis of intelligent decision-making system is a major research are in computer science and particularly artiﬁcial intelligence (AI). IAPTS perceives its environment and decides autonomously how to act in order to perform PT tasks as well as possible. AI-led cybersecurity solutions are often categorized under two types: expert-driven or automated systems utilizing unsupervised machine learning [16]. Expert-driven systems such AVs, FWs, IDPSs and SIEMs rely on security experts’ input and usually lead to high rates of errors until RL techniques were used to guide existence to more goal-directed learning systems that provide autonomous or semi-autonomous decision making which reﬂect the real-world context and especially offensive security domain such as vulnerabilities assessment and PT context [11]. The main reasons for our choice of RL are: • Effective autonomous learning and improving through constant interaction with the environment;

• Reward-based learning and existing ﬂexible rewarding schemes which might be delayed to enable RL agent to maximize a long-term goal;

• Richness of the RL environment which helps in capturing all major characteristics of PT including uncertainty and complexity.

1. Reinforcement learning is a variant of machine learning and a branch of AI. It allows software agents to automatically determine the ideal behavior within a speciﬁc context, in order to maximize its performance. Only simple feedback (reward) is required for the agent to learn how to adjust its behavior. In other words, it is based on an agent capable of learning and making decisions, and repeatedly interacting with its environment [5]. Interactions between the agent and the environment proceed by the agent observing the state of the environment, selecting an action which it believes is likely to be beneﬁcial, and then receiving feedback (reward) from the environment that provides an indication of the utility of the action [15]. As shown in Figure 2, RL allows an agent to learn from its own behavior within the RL environment by exploring it and learning how to act based on rewards received from its actions. This decision policy can be learned once and for all, or be improved or adapted if better results are encountered in the future. If the problem is appropriately modeled, some RL algorithms can converge to the global optimum which is the ideal behavior that maximizes the overall reward [5].



**Figure2. Reinforcementlearning(RL)agentobservesthestateoftheenvironment x(t) attimet,selects an action a(t) based on its action selection policy, and transitions to state x(t +1) at time t +1 and receives a reward r(t + 1). Over time, the agent learns to pursue actions that lead to the greatest cumulative reward [15].**

RL scheme excludes the need for signiﬁcant intervention from a human expert. In addition, less time is allocated for the learning and customization as it is the case with ML and expert systems, respectively. InadditiontowhathasbeensaidaboutthesuitabilityofRLforenhancingtheautomation of PT solutions, RL is a very active domain of research. Several new algorithms have been introduced recentlyalongwithsomeveryefﬁcienttoolboxesandimplementationswiththeabilitytosolvecomplex RL problems under constrained resources with great results [17,18].

* 1. Towards POMDP

Modeling of PT In PT, an attack is a set of tasks that are launched and executed, manually by a human tester or automatically by a PT platform, following a certain order in order to fulﬁll a goal or reach an objective. Depending on the context, the goal can be predeﬁned or unknown and also can vary throughout the attack. The goal (objective) of the attack is known within the PT community as the target which can be either logical or physical entity. Often, the target is a computer (physical or virtual machine with an operating system and applications) or a computer network or some information hosted on a computer such as ﬁles, database servers, or web servers. The attack target can also switch during an attack if a more valuable or easily exploitable target is identiﬁed to serve as a pivot later on. Furthermore, it is also common that an attack has no speciﬁc target, e.g., a script kiddie running a set of exploits against all reachable machines [10–12]. The starting point for this research is an automated PT system which lack for efﬁciency and optimization which in term of number of covered tests and the consumed resources and time as any PT test should not last forever and consume an excessive amount into performing or exploring irrelevant tests along with ensuring that no threat is ignored or underestimated. Therefore, the aim is developing an RL-led autonomous PT system that utilizes RL and other techniques at different levels of the practice to improve performance, efﬁciency, testing coverage and reliability [12]. The starting point was by adopting POMDP [19–25] for modeling PT as RL problem. POMDP models an agent that interacts with an uncertain environment. A POMDP can be deﬁned using the tuple M = <S, A, O, T, Ω, R, b1>. The sets S, A and O contain a ﬁnite number of states, actions and observations, respectively. The function T : S \* A \* S -> [0, 1] deﬁnes the stochastic state transitions. After executing action a ∈ A in state s ∈ S, the state changes stochastically to state s0 <S> with probability T(s, a, s0) = P(s0|s, a). The function R : S \* A -> R represents the reward function, such that the reward R(s, a) is received after executing action a <A> in state s <S>. The function Ω : A \* S \* O -> (0, 1) represents the observation function. Instead of observing the state s0 directly, the agent receives observation o <O> with probability pr(a, s0, o) = P(o|a, s0). The interaction between the agent and the environment is visualized in Figure 2. The agent executes action a, after which it receives observation o and reward R(s, a). Here it is important to note that it does not receive information about the state s itself. ThemodelingofPTasaPOMDPproblemandthejustiﬁcationofthechoicewasalreadydetailed in [12] and some highlights will be provided in the next sub-sections.

* 1. POMDP Solving Algorithm

RL algorithms are methods for solving real-world problems modeled in the form of MDPs or POMDPs which usually involve complex and sequences of decisions in which each decision affects what opportunities are available later. In this work, we are not concerned with the development of improvement of a new RL solving algorithm or methods, but only with ﬁnding the appropriate algorithm relevant to our problem which produces acceptable results [16]. When it comes to solving a large and complex RL problem is often complicated and therefore an adequate choice of the solving algorithms and approach should be made. For solving the PT POMDP complex environment, the IAPTS should rely on different solving algorithms rather than simply one. In fact, depending on the context, IAPTS will adapt to utilize to most adequate solving approach. Furthermore, the choice of different algorithms is justiﬁed by the constraints IAPTS may face in terms of the available resources (time, memory and computational) which make the use of one solving algorithm challenging, Thus adopting a ﬂexible approach where the accuracy is often sacriﬁced to acceptance. Finally, it is important to recall that large environments can also cause challenges when solving algorithms especially when dealing with a large number of transitions and observations or opting for a static reward scheme [16]. Most RL solving algorithms fall under two major categories: the reward (value) oriented solving and policy search solving [15]. The reward approach allows an RL agent evolving within the environment to select the sequences of actions that lead to maximizing the overall received reward or minimize received sanctions in the long term run and not only in the immediate future, this approach aims to dress an optimized and comprehensive rewarding function which relies on the atomic reward values associated with the RL environment to determine and an optimal (best possible) rewarding scheme (function) for each transition and observation. In terms of efﬁciency, this solving approachisoftencomplexandtime-consumingwithseveralcasesofaninﬁnitehorizoniftheproblem representation is not consistent enough and optimized [5]. The second approach, namely policy search seeks to construct a decision policy graph which is in practice done by learning the internal state/action mapping of the environment and using a direct search method for identifying policies that maximize the long term reward. The optimal policy is reachedwhenallthestatesandalltheactionsaretriedandallocatedasufﬁcientamountoftimetoﬁnd the best possible associated policies. In this research, we opted for the use of both reward-optimization and policy-search approaches. For the purpose of implementing policy-search algorithms, we found it is useful to include both on-policy and off-policy implementation to allow a better evaluation in terms of policies’ quality. The IAPTS POMDP solving module will use a powerful off-the-shelf POMDP-solverallowingtheuseofdifferentsolvingapproachesandstate-of-the-artalgorithmtoallow the exclusion of all external factors when it comes to evaluating the performances of different solving algorithms [18]. Initially, the following algorithms were shortlisted.

* 1. PERSEUS Algorithm

PERSEUS is a randomized point-based Value Iteration for POMDPs proposed by [15] that performs approximate value backup stages to ensure that, in each stage, the value of each point in the belief set is improved. The strength of this algorithm is its capacity of searching through the space of stochastic ﬁnite-state by performing policy-iteration alongside to the single backup which improves the value of the belief points. Perseus backs up also a random basis by selecting a subset of points within the belief set which is enough to improve the value of each belief point in the global set. In practice, PERSEUS is reputed to be very efﬁcient because of the approximate solving nature and is the best candidate for solving large size POMDP problems as it operates on a large belief set sampled by simulating decision sequences from the belief space leading to signiﬁcant acceleration in the solving process [19].

* 1. GIP Algorithm

GIP (generalized incremental pruning) is a variant of the POMDP exact solving algorithm family relying on incremental pruning. GIP algorithm replaces the LPs that we’re used in several exact POMDP solution methods to check for dominating vectors. GIP is mainly based on a Benders decomposition and uses only a small fraction of the constraints in the original LP. GIP was proven in [18] that it outperforms commonly used vector pruning algorithms for POMDPs and it reduces signiﬁcantly the overall running time and memory usage especially in large POMDP environment context. The latest version of GIP is, to the best of our knowledge, the fastest optimal pruning-based POMDP [17]. In this work we introduced some non-functional changes to the current implementation of GIP notably into the belief sampling to enable the use of an external belief rather than sampling the belief from POMDP environment at the beginning of the solving process, therefore the agent belief will be uploaded directly to the RL environment for efﬁcient use [17,18].

* 1. PEGASUS Algorithm

PEGASUS is a policy-search algorithm dedicated to solving large MDPs and POMDPs and was initially proposed by [20]. It adopts a different approach to the problem of searching a space of policies given a predeﬁned model as any MDP or POMDP is ﬁrst transformed into an equivalent POMDP in which all state transitions (given the current state and action) are deterministic and thus reducing the general problem of policy search to one in which only POMDPs with deterministic transitions are considered. Later, an estimation value of all policies is calculated making the policy search simply performed by searching for a policy with a high estimated value. This algorithm has already demonstrated huge potential as it produces a polynomial rather than exponential dependence on the horizon time making it an ideal candidate for the penetration testing POMDP solving [20].

* 1. Other Candidates

In addition to the candidates, other RL algorithms will be considered such as backwards induction and ﬁnite grid. The latter is an instance of point-based value iteration (PBVI) and will be mainly utilized in determining the shortest attack-path when more than one policy is found. Some of the proposed algorithms are already part of the POMDP-solver software and an optimized implementation is provided by the contributor and constantly improved over the versions. Nonetheless, some algorithm was implemented and integrated for the sake of benchmarking [21, 22]. Initially, and as the research focus was to dress a high-quality POMDP model representation for the PT practice bridging the gap between the theoretical research and real-world situation facing the industry professional, the use of such “ready solution” was highly recommended and was hopeful in advancing the research and also for the impact of the results obtained [23]. Finally, we evaluated the algorithm Palm leaf search (PLEASE) designed to solve POMDPs problems with large observation spaces [24].

* 1. Deep Reinforcement Learning Algorithms

Deep Reinforcement Learning methods are similar as original reinforcement learning algorithms but the agent uses deep neural networks.

Agents which use Deep Reinforcement learning methods have good performance to adapt to the environment.

Recently Deep Reinforcement learning methods are used to auto penetration testing and get good results.

[] and [] proposed deep q learning based auto penetration testing methods using NAS package and python tensorflow package.

They implemented deep q learning agents using tensorflow and tested it with NAS environments.

These researches showed good results in tiny scenarios however they didn’t show the result for big networks.

[] proposed A3C model based auto penetration testing methods and this is acceptable for post penetration testing.

It concentrated on how to reduce the training time on Deep Reinforcement learning based penetration testing.

Experiment shows that this method can reduce the time largely using parallel computation and also can be acceptable to post penetration testing.

Appendix B: Data and Programs Used for this Report[[5]](#footnote-5)

*(required section and optional section)*

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) = 195

#

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

    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: [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

    services: [ssh]

    processes: [tomcat]

# two row 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

Required

To ensure that research outcomes can be validated:

* all data and/or programs which you created and/or gathered and used for the results presented in this document must be submitted;
* a softcopy of the data and/or programs must also be provided;
* if the individual file size exceeds 500 MB, please contact the Unit Coordinator on how to submit your files;
* Ensure that you use logical filenames which directly relate to a section, table or figure number or name of this document.
* If you used data and/or programs from by 3rd parties (e.g. from open access databases), you must include links to all sources used in a separate pdf document. These sources must be freely accessible to your first supervisor;
* All open access sources and databases used must also be referenced in the References section of this document;

Optional

In addition, you may include items in this appendix if that is relevant. If in doubt, please discuss with your supervisor(s).

Appendix C: Figures[[6]](#footnote-6)

*(required)*

Include in this appendix all figures used for the paper section of this document and any additional figures that may be relevant. If in doubt, ask your supervisor(s).

* Figures included in this section should have separate figure numbers from the figures used in the paper section (continue numbering your figures throughout this document) and be included in the List of Figures at the start of this document;
* Figures included in this section would normally be at a larger scale than the figures in the paper section;
* Figures should be in colour, however the information must be unambiguous in black and white (use e.g. different line types to differentiate information);
* The title of the figure should not be included in the image, but be typed below the figure;
* The title of the figure should be “Fig.” followed by a number and the text labelling the figure;
* All data and programs which were used to create figures of this document must be submitted in Learnline, refer to Appendix B for further instructions.

Appendix D: Tables[[7]](#footnote-7)

*(required if you use tables in this document)*

Include in this appendix all tables used for the paper section of this document and any additional tables that may be relevant. If in doubt, ask your supervisor(s).

* Tables included in this section should have separate table numbers from the figures used in the paper section (continue numbering your tables throughout this document) and be included in the List of Tables at the start of this document.
* Tables included in this section would normally be at a larger scale than the tables in the paper section;
* Tables should not use colour;
* The title of the table should not be included in the image, but be typed above the table;
* The title of the table should be “Table” followed by a number and the text labelling the table;
* All data and programs which were used to create tables of this document must be submitted in Learnline, refer to Appendix B for further instructions

References[[8]](#footnote-8)

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

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

[3] S. Jimenez, T. De-la-rosa, S. Fernandez, F. Fernandez and D. Borrajo, A review of machine learning for automated planning. The Knowledge Engineering Review, Vol. 00:0, pp. 124. 2009. [4] Y. Andrew and M. Jordan, PEGASUS: A policy search method for large MDPs and POMDPs. 16th Conf. on Uncertainty in Artificial Intel., 2013.

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

[6] 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.

[7] 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.

[8] 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.

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

[]. 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).

[2]. 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 2016 .

[3]. Applebaum, A.; Miller, D.; Strom, B.; Korban, C.; Wol, R. Intelligent, automated red team emulation. InProceedingsofthe32ndAnnualConferenceonComputerSecurityApplications(ACSAC’16),LosAngeles, CA, USA, 5–8 December 2016; pp. 363–373.

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

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

[6]. 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.

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

[8]. 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.

[9]. 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.

[10]. 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).

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

[12]. 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.

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

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

[3]. 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

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

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

[6]. J.Hoffmann,Simulatedpenetrationtesting: FromDijkstratoaaTuringTest++. 25thInt. Conf. onAutomated731 Planning and Scheduling, 2015, AAAI Press.732

[7]. C.Sarraute,Automatedattackplanning. InstitutoTecnologicodeBuenos-aires,Ph.D.Thesis,2012,Argentina.733

[8]. 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

[9]. 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

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

[11]. 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

[12]. 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

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

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

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

[16]. 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

[17]. 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

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

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

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

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

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

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

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

[5] D. of Industry, “$50 million investment into cyber security research and industry solutions,” Ministers for the Department of Industry, Innovation and Science ​ , 22-Sep-2017. [Online]. Available: http://minister.industry.gov.au/ministers/craiglaundy/media-releases/50-million-investment-cyber-se curity-research-and-industry​. [Accessed: 13-Mar-2018].

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

1. List here all figures you have used in the paper and other sections of this document, number all figures, remove this footnote [↑](#footnote-ref-1)
2. List here all tables you have used in the paper and other sections of this document, number all tables, remove this footnote [↑](#footnote-ref-2)
3. List here all abbreviations used in the paper and the other sections of this document, remove this footnote [↑](#footnote-ref-3)
4. List here all variables and their units used in the paper and other sections of this document, remove this footnote [↑](#footnote-ref-4)
5. Amend headings and text as needed, remove this footnote [↑](#footnote-ref-5)
6. Amend headings and text as needed, remove this footnote [↑](#footnote-ref-6)
7. Amend headings and text as needed, remove this footnote [↑](#footnote-ref-7)
8. It is highly recommended that you use EndNote to make the list of references for this document, remove this footnote [↑](#footnote-ref-8)