Vahur Varris

USING ADVERSARIAL REINFORCEMENT LEARNING TO EVALUATE THE IMMUNE RISK-ASSESSMENT

Professional thesis report

Company: IABG mbH

EURECOM advisor: Melek Önen, PhD

Industrial advisors: Martin Riedl, PhD

Sebastian Belkner

Philip Trautmann, PhD

Non-Confidential

Digital Security track

2020

Abstract

During the recent years, as the world becomes more digitalized, there has been a rise in cybercrime and rise of activity from stealthy threat actors such as advanced persistent threats, which poses a challenge for many companies to effectively secure their networks. The goal of the IMMUNE project is to focus on the security of modern industrial networks with an aim on self-defending resilient networks that use the modern networking paradigm software-defined networking.

The internship investigates the feasibility of using reinforcement learning, which has enjoyed many breakthroughs in the past few years, in an adversarial setting. The main idea is to simulate 2 reinforcement learning agents: the attacker and the defender in a same partially observable network environment where they compete against each other with a goal for the network defender to learn the best countermeasures in each stage of the attack and therefore contribute towards the larger goal of the project of enabling autonomous network self-deference.

As a result, a novel proof-of-concept reinforcement learning approach is devised, that allows defender to learn a policy of best reconfigurations provided by software-defined networking paradigm to take into account the given situational picture provided by the sensors in the network and information from risk assessment system that detects anomalous behaviour. This approach allows to take into account the known vulnerabilities in the system and devise countermeasures based on the behaviour they cause to the system.

Résumé

Au cours des dernières années, alors que le monde se numérise de plus en plus, on a assisté à une augmentation de la cybercriminalité et à un accroissement de l'activité des acteurs de menaces furtives telles que les Advanced Persistent Threat (APT), ce qui pose un défi à de nombreuses entreprises pour sécuriser efficacement leurs réseaux. L'objectif du projet IMMUNE est de se concentrer sur la sécurité des réseaux industriels modernes dans le but de mettre en place des réseaux autodéfendables et résistants qui utilisent le paradigme moderne de la mise Software-Defined Networking (SDN).

Le stage étudie la faisabilité de l'utilisation de l'apprentissage par renforcement, qui a connu de nombreuses percées ces dernières années, dans un cadre contradictoire. L'idée principale est de simuler deux agents d'apprentissage par renforcement: l'attaquant et le défenseur dans un même environnement de réseau partiellement observable où ils sont en compétition l'un contre l'autre avec pour objectif que le défenseur du réseau apprenne les meilleures contre-mesures à chaque étape de l'attaque et contribue ainsi à l'objectif plus large du projet qui est de permettre l'autodéfense autonome du réseau.

En conséquence, une nouvelle approche d'apprentissage du renforcement de la preuve de concept est conçue, qui permet au défenseur d'apprendre une politique des meilleures reconfigurations fournies par un paradigme de réseau défini par logiciel pour prendre en compte l'image donnée de la situation fournie par les capteurs du réseau et les informations du système d'évaluation des risques qui détecte les comportements anormaux. Cette approche permet de prendre en compte les vulnérabilités connues du système et de concevoir des contre-mesures basées sur le comportement qu'elles entraînent pour le système.

Executive Summary

The internship took part between March and August 2020 in the Predictive Modelling Department (IZ60) in the context of the IMMUNE project, which aims to develop a resilient and self-defending industrial network using the novel paradigm called software-defined network. The IMMUNE project is conducted by a consortium of companies and institutions including IABG, Airbus, Fraunhofer, IFAK, Siemens and University of Hamburg.

The IABG IMMUNE team is responsible for developing the tools to monitor and model the network's risk. After the team developed the tools to model the risk of the network, there was an idea to use this risk metric as an input for the agent who can effectively react to lower that risk score in case of attacks to the system. This internship investigated the possibilities and feasibility of using reinforcement learning agents in an adversarial setting, where attacker and defender are simulated by intelligent agents in a network. The main interest is whether the network risk scores are sufficient to provide defender with accurate situational picture, so the agent is able to learn effective countermeasures from that information.

Reinforcement learning is a machine learning technique, where an agent is simulated in an environment where it learns the behaviour only based on the given reward. There have been many breakthroughs in this field over the past years, mostly by combining the traditional methods with deep learning techniques.

During the internship, a framework was developed using the IMMUNE network models, that simulates the attacker by taking into account all the possible attack paths in the network, that are derived from all currently known vulnerabilities, and the defender whose goal is to reduce the risk to the network. The attacker's actions cause unusual behaviour in the network sensors, which are used as an input to calculate the current risk metrics. The defender receives only this information and needs to learn which countermeasures to take in order to minimize the network's risk. The results of the

experiments showed that by using the risk score as a part of defender's reward function, it was able to learn to select countermeasures that reduce the overall risk of the network.

There has been previous works related using adversarial reinforcement learning in a cyber security setting, but to the best knowledge of the author, there has not been a similar setup where an attacker is simulated using all the possible attack paths and defender needs to react based on aggregated information from the risk assessment system by changing the actual topology of the network.

Table of contents

Abstract 2
Résumé 3
Executive Summary 4
Table of contents 6
List of figures 8
List of tables 9
1 Introduction 10
1.1 Company 10
1.2 Project 10
1.3 Internship objectives 11
2 Theoretical foundations 12
2.1 Reinforcement Learning 12
2.1.1 Markov Decision Process 13
2.1.2 Fundamentals of Reinforcement Learning 13
2.1.3 Algorithms Implemented 17
2.1.4 Known Issues with Reinforcement Learning 22
2.2 Multi-Agent Reinforcement Learning 23
2.3 Software-Defined Networking (SDN) 23
3 IMMUNE project overview 26
3.1 IMMUNE models 26
3.1.1 IMMUNE model (IMM) 26
3.1.2 IMMUNE Risk Assessment (IRA) 28
3.1.3 Attack Graph (AG) 29
3.2 IMMUNE processes 30
3.2.1 Network model construction 31
3.2.2 Risk Analysis Pipeline and Reinforcement Learning Cycle 32
4 IMMUNE Reinforcement Learning 33
4.1 Overview 33
4.2 Previous Work with Reinforcement Learning in Cyber Defence 33

4.3 Environment Setup 34
4.4 Attacker 35
4.5 Defender 37
4.6 Experiments Setup 38
4.7 Experiments Results 42
4.7.1 Proof of Learning 42
4.7.2 Learned Policy Interpretation 43
4.7.3 Simulation 47
4.8 Discussion of Results 50
4.9 Further Work and Possible Improvements 51
4.10 Knowledge Transfer and Project Handover 51
5 Conclusions 53
References 55

List of figures

Figure 1. Taxonomy of Machine Learning 13
Figure 2. Reinforcement Learning process 14
Figure 3. Actor-Critic architecture [10] 21
Figure 4. SDN architecture [14] 24
Figure 5. Main IMMUNE network model 27
Figure 6. Simplified IMMUNE network model 28
Figure 7. Attack graph of the simplified IMMUNE network 30
Figure 8. IMMUNE processes 31
Figure 9. Agents Network Setup 35
Figure 10. Network configuration model1 39
Figure 11. Attack graph model1 40
Figure 12. Attack graph model2 41
Figure 13. Attack graph model3 41
Figure 14. Q-learning steps 43
Figure 15. Q-learning rewards 43
Figure 16. Scenario - Step 0 44
Figure 17. Scenario - Step 1 44
Figure 18. Scenario - Step 2 45
Figure 19. Scenario - Step 3 45
Figure 20. Scenario - Step 4 46
Figure 21. Configurations' risk 47
Figure 22. Steps Heatmap 48
Figure 23. Defender Reward Heatmap 49
Figure 24. Attacker Reward Heatmap 50

List of tables

Table 1. Example temporal difference state-action value representation 17
Table 2. Attacker State and Actions.....36
Table 3. Defender state and actions 37

1 Introduction

In this paragraph, the brief overview is given on the company, the project in which context the internship work is carried out, and the motivation and goals behind the internship.

1.1 Company

The internship is carried in a company Industrieanlagen-Betriebsgesellschaft mbH (abbreviated as IABG) located in Ottobrunn, Germany, on the outskirts of the Bavarian state capital Munich. The company was founded in 1961 by the West-German federal government as a central analysis and testing facility for the Ministry of Defence and the aeronautical industry. Since 1993, the company has been fully privatized and over time many additional business areas have added. IABG is active in fields such as:

- Automotive
- InfoCom
- Defence & Security
- Aeronautics and space
- Energy. [1]

IABG is a partner for EURECOM and an industrial member of the EURECOM consortium. [2]

1.2 Project

The internship is carried out in the context of applied research project IMMUNE which aims to develop and implement resilient and self-defending industrial network for next generation industry 4.0 factories and uses Airbus' factory of the future as a case study. The project is focused on using software-defined networking technology and distributed

systems as main underlying technologies. The project focuses on a variety of stages of cyber defence such as attack mitigation, detection and treatment. [3]

The project is carried out by a consortium of both industrial and academic partners and is funded by a German Ministry for Economic Affairs and Energy. [4]

1.3 Internship objectives

IABGs role in the IMMUNE consortium is to mainly work on network monitoring, intrusion detection and risk assessment. Before the start of the internship, there are already tools that have been developed by the IMMUNE team in IABG to model the network risk and attacker lateral movement using attack graphs.

The main goal of the internship is to implement and analyse effectives the usage of reinforcement learning in context of IMMUNE project for minimizing the network defender's risk and evaluate the feasibility of reinforcement learning for network self-defence.

The setting and initial starting point of the internship is to apply reinforcement learning in an adversarial setting, where both attacker and defender have limited information about the network and other actors. Defender is able to observe only aggregated, processed information (such as the output of a risk assessment system). Meanwhile the attacker tries to exploit system flaws to gain reward while the defender – equipped with a number of defence strategies – tries to mitigate costs.

Goals and objectives of the internship are as follow:

- Familiarize with the current state of the project.
- Comprehend literature concerning reinforcement learning.
- Implement and evaluate reinforcement learning in the IMMUNE project setting for autonomous self-defence to minimize the defender's risk.
- Assure the knowledge transfer to the project team and the company.

2 Theoretical foundations

This chapter gives an overview of the theoretical foundations behind reinforcement learning and the main concepts for technologies used in IMMUNE project such as software-defined networking.

2.1 Reinforcement Learning

This section gives a brief introduction to the basics of reinforcement learning and the algorithms implemented during the internship.

Reinforcement learning is a subfield of machine learning where instead of learning from a predefined dataset, an agent must learn behaviour through trial and error from a dynamic environment while maximizing its numerical reward signal. The agent is not aware of which actions return the maximum long-term reward and it has to learn it over time. The rewards can be immediate or subsequent, hence, the agent must be able to estimate the future consequences of current actions. [5, pp. 1-2]

One way for classifying the taxonomy of machine learning is presented in figure 1. Supervised learning represents classical machine learning where the labelled dataset is provided, and the task is to find a function to fit the input to the labels as either regression for continuous label or classification for discrete ones. Unsupervised machine learning is performed when the given data is unlabelled, and the algorithm's task is to find patterns in the underlying dataset, usually through classification, but other methods such as dimensionality reduction or anomaly detection are possible. Reinforcement learning, on the other hand, does not require any training data set, however the agent must be provided with rewards to make the learning possible.

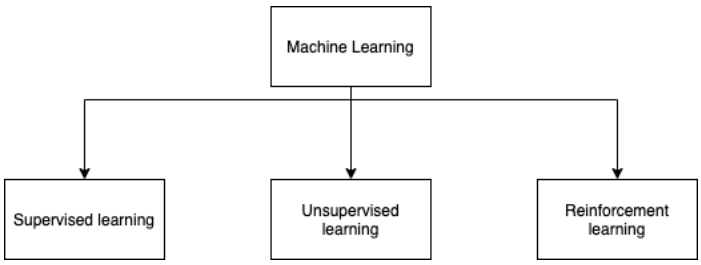

Figure 1. Taxonomy of Machine Learning

2.1.1 Markov Decision Process

Reinforcement learning environment is modelled as a Markov decision process (MDP) [6] which consists of 5 components:

- Set of states: S
- Set of actions: A – can be both discrete or continuous
- Transition probability function – given state s and next state s' and action a, returns the probability of transition: $P(s'|s,a)$
- Reward function –given state s and next state s' and action a, returns the numeric reward: $R(a,s,s')$
- Discounting factor for future rewards – $\gamma \in (0:1)$

2.1.2 Fundamentals of Reinforcement Learning

The main components in a reinforcement learning process are:

- Agent
- Environment
- Actions

During the learning process the agent interacts with the environment by taking an action during each timestep. The environment, which represents everything external to the agent, gives back a numeric reward and the new state of the environment. The goal of the agent is to maximize the rewards by learning which actions result in maximum future rewards

given the current state of the environment. The reinforcement learning process is visualized in figure 2. [5, pp. 47-48]

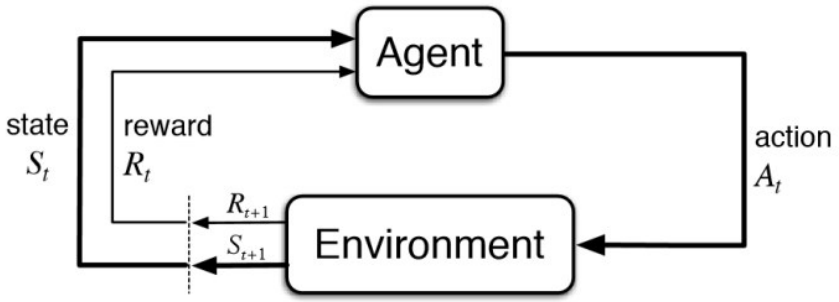

Figure 2. Reinforcement Learning process

The learning process happens during the sequence of time steps $t = 0 \ldots T$ (where T is the terminal/final state). The sequence $S_0, A_0, R_0, S_1, A_1, R_1 \ldots S_T, A_T, R_T$ is called an episode. [5, p. 48]

2.1.2.1 Policy

The main goal of reinforcement learning is to learn a policy which defines the agent's behaviour during each time step [5, p. 58]. Policy π is a function that maps agents actions defined as follows:

$$\pi(s) \rightarrow a$$

The policy can be deterministic or stochastic (probabilistic). The goal of the learning process is to learn the optimal policy which returns the action that maximizes the future rewards:

$$\pi_*(s) \rightarrow a$$

2.1.2.2 Return

As already mentioned before, the agent's goal is to maximize the future reward. The return *G* in the simplest case is the sum of all the rewards:

$$G = R_1 + \ldots + R_T$$

However, the issue with cumulative return is that for continuous tasks where T is approaching infinity, the return would also approach infinity. Furthermore, the immediate

and short-term rewards are more valuable than larger rewards in the future, hence the idea of discounting the future rewards with γ:

$$G = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

2.1.2.3 Value

The value estimates for the agent how good it is to be in a given state [5, p. 58]. More formally, it is a function that takes a state as an input under a given policy and gives back the expected return:

$$V_s = E_\pi[G_T | S_t = s] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s]$$

2.1.2.4 Q- value

The value of taking some action while being in a given state is called the Q-value or alternatively state-action value [5, p. 58]. Formally noted as:

$$Q_\pi(s, a) = E_\pi[G_T | S_t = s, A_t = a] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a]$$

2.1.2.5 On-policy and Off-policy

There exist 2 types of reinforcement learning algorithms: on-policy and off-policy. On-policy methods attempt to evaluate or improve the policy that is used to make decisions, whereas off-policy methods evaluate or improve a policy different from the one that is used to generate the data. [5, p. 100]

2.1.2.6 Model based and model-free

Another way to distinguish algorithms is whether they are model-based or model-free. In model-based reinforcement learning the agent's goal is to learn the model of the environment and to solve the task via planning. However, as the task is difficult for larger environments, in practice most used methods are model-free where the agent learns via trial and error and does not have access to the full model of the environment. [5, p. 7]

2.1.2.7 Temporal Difference Learning Methods

Temporal difference learning methods are model-free reinforcement learning paradigms, where the main idea is to bootstrap the intermediate estimates. During each step in the environment, the agent immediately updates the target value from the received reward:

$$V(S_t) \leftarrow V(S_t) + \alpha\,[R_{t+1} + \gamma\, V(S_{t+1}) - V(S_t)]$$

where hyperparameters α is the learning rate ($0 \leq \alpha \leq 1$) and γ is the discount factor of the future rewards (also $0 \leq \gamma \leq 1$). [5, p. 120]

There is a trade-off with temporal difference methods between the agent exploring the environment versus the agent exploiting (taking the currently known best action). On the one hand, after too little exploitation, there is a danger to get stuck in a local maximum, on the other hand, when exploring too much, the algorithm may not converge or converge slower. The epsilon-greedy algorithm [5, p. 26] is commonly used in practice. The ε parameter ($0 \leq \varepsilon \leq 1$) represents the probability of the agent exploring versus exploiting. The decaying epsilon strategy can be also used where during the first episodes the epsilon value is close to one and decreases as the training process proceeds.

2.1.2.8 Policy Gradient Learning Methods

In contrast to the temporal difference learning where the main unit in the learning process is the value function that is being updated via state-action pairs, the policy gradient methods focus on directly optimizing the policy itself. The paradigm involves policy parameter (noted as θ) and parameterized policy (noted π_θ). During the training the function $J(\theta_t)$ is used to measure the performance of the policy parameter as a loss function. [5, p. 321]

During the learning process, the goal is to maximize the policy parameter with regards to the loss function using the gradient ascent algorithm noted formally as:

$$\theta_{t+1} = \theta_t + \alpha \,\forall\, J(\theta_t)$$

Where α is the learning rate ($0 \leq \alpha \leq 1$) and $\forall\, J(\theta_t)$ is an estimate which expectation approximates the gradient of the performance measure with respect to θ_t. [5, p. 321]

2.1.3 Algorithms Implemented

During the internship, the following reinforcement learning algorithms were implemented:

- Q-Learning
- Deep Q-learning
- Advantage Actor Critic (A2C)

The first 2 are examples of temporal difference learning and the Advantage Actor Critic algorithm belongs to the policy gradient methods family.

2.1.3.1 Q-learning

Q-learning [7] is a model-free off-policy reinforcement learning technique using the temporal difference learning concept and is defined as:

$$Q(S_t, A_{t,}) \leftarrow Q(S_t, A_{t,}) + \alpha\,[R_{t+1} + \gamma\, max_a Q(S_{t+1}, a) - Q(S_t, A_{t,})]$$

Hyperparameter α is the learning rate ($0 \leq \alpha \leq 1$) and γ is the discount factor of the future rewards (also $0 \leq \gamma \leq 1$).

Q-learning is an off-policy algorithm which means that during the update step of the Q-value, the maximum action value for the next state is used instead of the action provided by the current policy.

The Q-table with state-action values are stored in the memory in a tabular format during the training process, which means that the table size is $S \times A$, making the Q-learning infeasible for problems with large action and/or state spaces. Example tabular representation is presented in table 1 below.

	Action 1	Action 2
State 1	Q(state 1, Action 1)	Q(state 2, Action 2)
State 2	Q(state 2, Action 2)	Q(state 2, Action 2)

Table 1. Example temporal difference state-action value representation

Pseudocode for the Q-learning algorithm is as follows:

1. Initialize Q-table
2. Loop until terminal state s or maximum steps are reached
    a. Observe state s
    b. Select action *a* based on ε-greedy algorithm
        i. Generate random value r
        ii. If $r > \varepsilon$-parameter: $a \leftarrow random(A)$
        iii. Otherwise $a \leftarrow argmax_a Q(s)$
    c. Take action *a*
    d. Receive reward r and next state s' from the environment
    e. Update Q-table:
$$Q(s,a) \leftarrow Q(s,a) + \alpha\,[r + \gamma\, max_a Q(s') - Q(s,a)]$$
    f. Set s = s' as current state

2.1.3.2 Deep Q-learning

As mentioned in the previous chapter, traditional Q-learning has issues with large action-state spaces. Researchers from Deepmind proposed a deep Q-network [8] that uses deep learning and replaces Q-table with a deep neural network that approximates the Q-table to address that issue. The goal of the Deep Q-learning is to train approximator θ such that:

$$Q(S,A,\theta) \approx Q(S,A)$$

with a loss function:

$$L(\theta) = E[(r + \gamma\, max_a Q(s',a',\theta')) - Q(s,a,\theta))^2]$$

Where θ' represents the parameters from the previous iteration. In [8], the rolling replay buffer is used to store the states, actions, rewards, and next states and during the learning

process minibatch is randomly sampled from the buffer and the network is trained with stochastic gradient descent.

In the context of the internship, the Deep Q-network was implemented not because of the infeasibility of Q-learning for the given problem, but to compare the performance of deep learning-based Q-learning to other algorithms. Furthermore, similar to the researchers in Deepmind who implemented the Deep Q-network using a technique called experience replay, which stores the previous states-action-rewards transitions in a buffer in the memory, and then during the training, it samples a random minibatch from there. During the internship, 2 different implementations were made of Deep Q-network, one using the replay buffer, and the other without it to compare the approach.

The pseudocode for the Deep Q-learning algorithm with replay buffer is as follows:

1. Initialize replay buffer B and Q-network with weights θ
2. Loop until terminal state or maximum steps are reached
    a. Observe state s
    b. Select action a
        i. Generate random value r
        ii. If $r > \varepsilon$-parameter: $a \leftarrow random(A)$
        iii. Otherwise $a \leftarrow argmax_a Q(s, \theta)$
    c. Take action a
    d. Receive reward r and next state s' from the environment
    e. Store the following tuple to replay buffer B: (s, a, r, s')
    f. Sample set of random transitions (s, a, r, s') from replay buffer B and calculate target value for each of the transitions:
        i. If s' is terminal state: $target \leftarrow r$
        ii. Otherwise: $target \leftarrow r + \gamma\, max_a Q(s', \theta)$

iii. Adjust the weight parameter θ using gradient descent method of the Q-network with a L2 loss function: $(target - Q(s,a))^2$

The pseudocode for the Deep Q-learning algorithm without replay buffer is as follows:

1. Initialize Q-network with weights θ
2. Loop until terminal state or maximum steps are reached
    a. Observe state s
    b. Select action a
        i. Generate random value r
        ii. If $r > \varepsilon$-parameter: $a \leftarrow random(A)$
        iii. Otherwise $a \leftarrow argmax_a Q(s,\theta)$
    c. Take action a
    d. Receive reward r and next state s' from the environment
    e. Compute target
        i. If s' is terminal state: $target \leftarrow r$
        ii. Otherwise: $target \leftarrow r + \gamma\, max_a Q(s')$
        iii. Adjust the weight parameter θ using gradient descent method of the Q network with a L2 loss function: $(target - Q(s,a))^2$

2.1.3.3 Advantage Actor Critic (A2C)

Advantage Actor Critic [9] method is policy gradient method used in an actor-critic setting. As discussed in the section on policy gradient methods, the main idea is to optimize the policy directly. An actor-critic setting is an interesting hybrid approach where two models are used together (visualized in a figure 3): an actor model that learns

the policy (as probability distribution of actions) and a critic model, that learns the value function. During the training phase, the advantage operator A is used which gives the actor (policy) model feedback on how well the policy is performing, and in which direction the actor should adjust it. Formally, the actor model updates the policy parameter (noted as θ) and parameterized policy (noted π_θ) and the critic model the value parameter θ_v which tracks the state-value function $V(s, \theta_v)$. The beforementioned advantage operator is defined as follows:

$$A(a_t, s_t) = Q(a_t, s_t, \theta) - V(s_t, \theta_v),$$

where Q and V functions are estimated by their respective models.

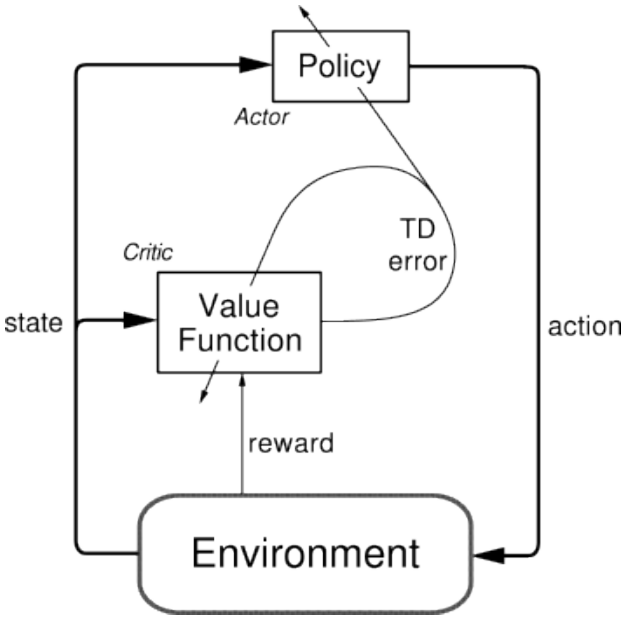

Figure 3. Actor-Critic architecture [10]

The pseudocode for the Advantage Actor Critic algorithm is as follows:

1. Initialize actor network θ and critic network θ_v
2. Loop until terminal state or maximum steps are reached
    a. Observe state s
    b. Select action a according to actor model:
        i. Receive categorical distribution from the policy model: $distribution \leftarrow \pi_\theta(s)$

ii. Sample action from distribution $a \leftarrow distribution.sample()$

iii. Receive the log-probability of the action from distribution $p \leftarrow distribution(a)$

c. Compute the state s value v from critic model: $v \leftarrow V(s, \theta_v)$

d. Take action a

e. Receive reward r and next state s' from the environment

f. Store (p v, r) to buffer

3. After the episode is done:

a. Compute the actual discounted returns for each timestep from the buffer:

i. $R \leftarrow 0$

ii. for each (p, v, r) from the end of the buffer to beginning

1. Compute actual discounted return: $R \leftarrow r + \gamma R$

2. Compute advantage: $A \leftarrow R - v$

3. Train the actor with a loss function: $-\mathrm{p} * \mathrm{A}$

4. Train the critic with a L2 loss function: $(r - v)^2$

2.1.4 Known Issues with Reinforcement Learning

Although reinforcement learning has enjoyed many breakthroughs during the past few years, mainly due to combining older learning techniques with deep neural networks. Great contribution to the field of reinforcement learning was the event in 2016 where research company DeepMind developed an algorithm branded as AlphaGo that managed to defeat the world champion in an ancient Chinese board game Go. However, there are still many unsolved issues with reinforcement learning, most of which become especially problematic when applied to real-life environments, out of toy simulations and games.

The biggest issues [11] can be generalized between efficiency and reward engineering and safety.

Efficiency in reinforcement learning is a major issue, as the learning happens online and the agent needs to generate its own training data, in many cases the episodes take long time to play and the agent, usually, only obtains the reward in the end of the episode and needs to generalize it over the whole trajectory of actions taken. As already mentioned, when discussing Q-learning, classical tabular learning methods suffer from Bellman's curse of dimensionality, while deep learning methods help to manage this issue, they introduce the black-box models in the form of neural networks, making the model less interpretable.

The second big issue with reinforcement learning is reward engineering and accompanying potential safety issues. In supervised learning the data is labelled and some standard algorithm such as mean squared error is used to calculate the loss and the goal of the learning is to minimize that loss. In reinforcement learning, the reward function has to be manually set for each learning task. Furthermore, as the agent's goal is to maximize the reward, it can learn unintended ways to maximize the reward, while not actually learning to solve the intended problem, which in real-life scenarios can bring about potential safety issues.

2.2 Multi-Agent Reinforcement Learning

Reinforcement learning can be applied to the environment where multiple agents operate together in the same environment and depending on the goals of the agents, this setting can be either cooperative, competitive or mixed, based on the agent's given reward strategy. Typically, in cooperative tasks the agents share the reward function, and both get positive reward when accomplishing a common goal. In the competitive task, the rewards are typically zero-sum, which means that one agent's positive reward is symmetrical to the other's negative. [12]

2.3 Software-Defined Networking (SDN)

The IMMUNE project takes advantage of networking technology called software-defined networking (SDN), which is a paradigm that reinvents the management of the network

by splitting up the control and forwarding functionality into control and data planes (figure 4). Although this internship does not focus on the technicalities of software-defined networks, it is crucial to understand the capabilities offered by this concept, in order to model the capabilities of the network defender for reinforcement learning as the IMMUNE project is built on top of it.

In traditional network architectures, both control and forwarding logic is implemented in the network devices itself, meaning that for especially large multi-vendor networks the maintenance is very expensive and prone to errors due to misconfiguration. On the other hand, as SDN separates the routing and forwarding decisions of networking elements (routers, switches, and access points) from the data plane, the network administration and management becomes less complicated because the control plane only deals with the information related to logical network topology, the routing of traffic. In contrast, the data plane orchestrates the network traffic in accordance with the established configuration in the control plane which is centralized in a controller that dictates the network policies. [13]

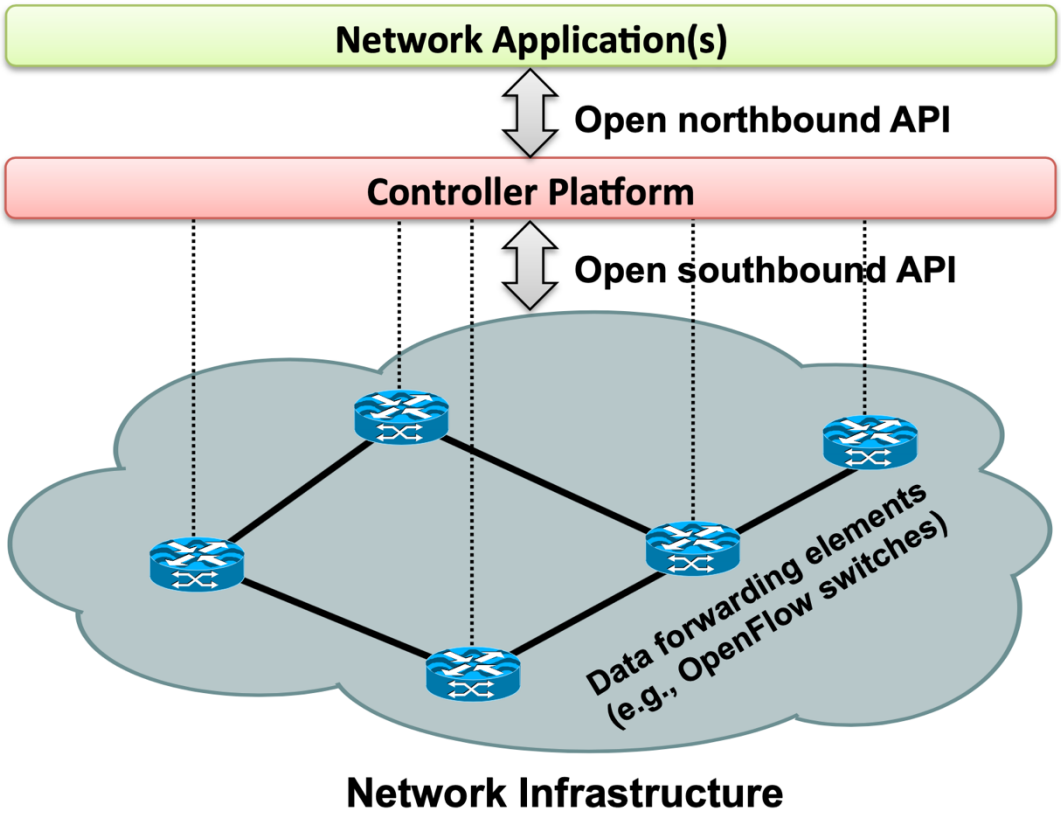

Figure 4. SDN architecture [14]

In the context of this internship, the main functionality of interest that software-defined networking brings to the table is the central management of the network, which allows to effectively reconfigure the network and therefore isolate or disconnect hosts or subnets that are infected or under attack. Since the IMMUNE consortium is not yet as far with the project to have a live running instance of the software-defined network, the model that mimics the same functionality is used instead.

3 IMMUNE project overview

This chapter describes the working blocks of the IMMUNE project and gives more detailed description in which context the reinforcement learning is applied and what are all the working blocks it has to work together or relies upon.

3.1 IMMUNE models

The IMMUNE team also uses different types of which represent different types of models and graphs to carry out various tasks. These data structures with their abbreviations are as follows:

- IMMUNE model (IMM)
- IMMUNE Risk assessment (IRA)
- Attack graph (AG)

3.1.1 IMMUNE model (IMM)

IMMUNE model is an object-oriented representation of industrial network. It consists of following parts:

- Hosts
- Routers
- Sensors

The hosts represent devices in the network. They can be either PCs, servers or programmable logic controllers (PLCs). The hosts have network interfaces and IP addresses. The IMMUNE model also describes the current software that is installed on a host and which ports (if any) the software is using. The software is described using the Common Platform Enumeration (CPE) which is a standard developed by National Institute of Standards and Technology (NIST) [15].

The routers represent the networking infrastructure with different IP address ranges and subnets. The routers also model the network firewall rules; therefore, they contain the information that describes which hosts and which parts of the network can communicate with each other and whether the ports are open.

The sensors that network contain are meant to model the host-based intrusion detection system, so if the host will be compromised or exhibits anomalous behaviour, then the sensors can detect it with some probability, on the other hand, the sensor are modelled in a way that false negatives are also present, so intrusion can go undetected.

The main IMMUNE network model is developed by the consortium to imitate the industrial network. It has a large multi-segment network with many different hosts, industrial devices and deployed sensors.

The main model is visualized in a figure below, with yellow nodes as industrial devices, green nodes as sensors, dark blue nodes as routers in a network and blue as hosts in the network.

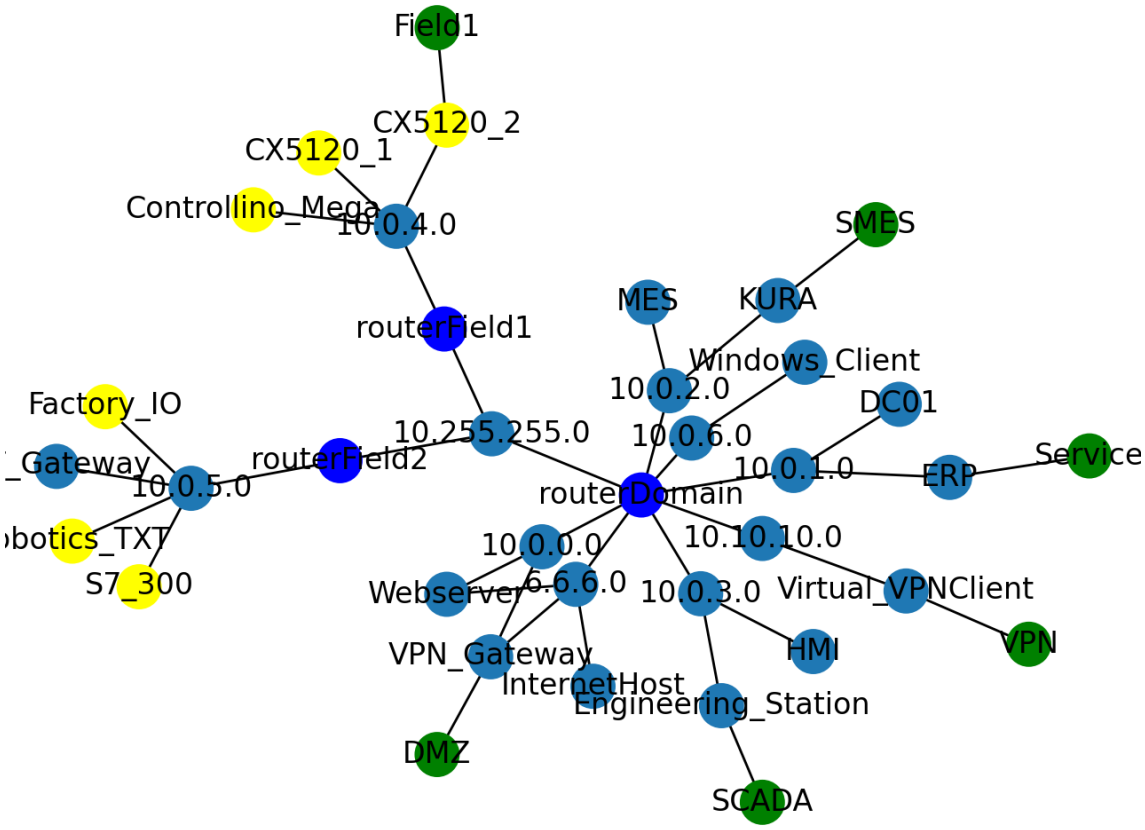

Figure 5. Main IMMUNE network model

The issue with the large network model is that it is much more complex to use to validate the initial approach for the reinforcement learning, instead a simplified network model is used for the reinforcement learning task which has only 5 hosts: Attacker, Webserver, SSH, FTP and DB, is used for that. The simplified model is illustrated in a figure below.

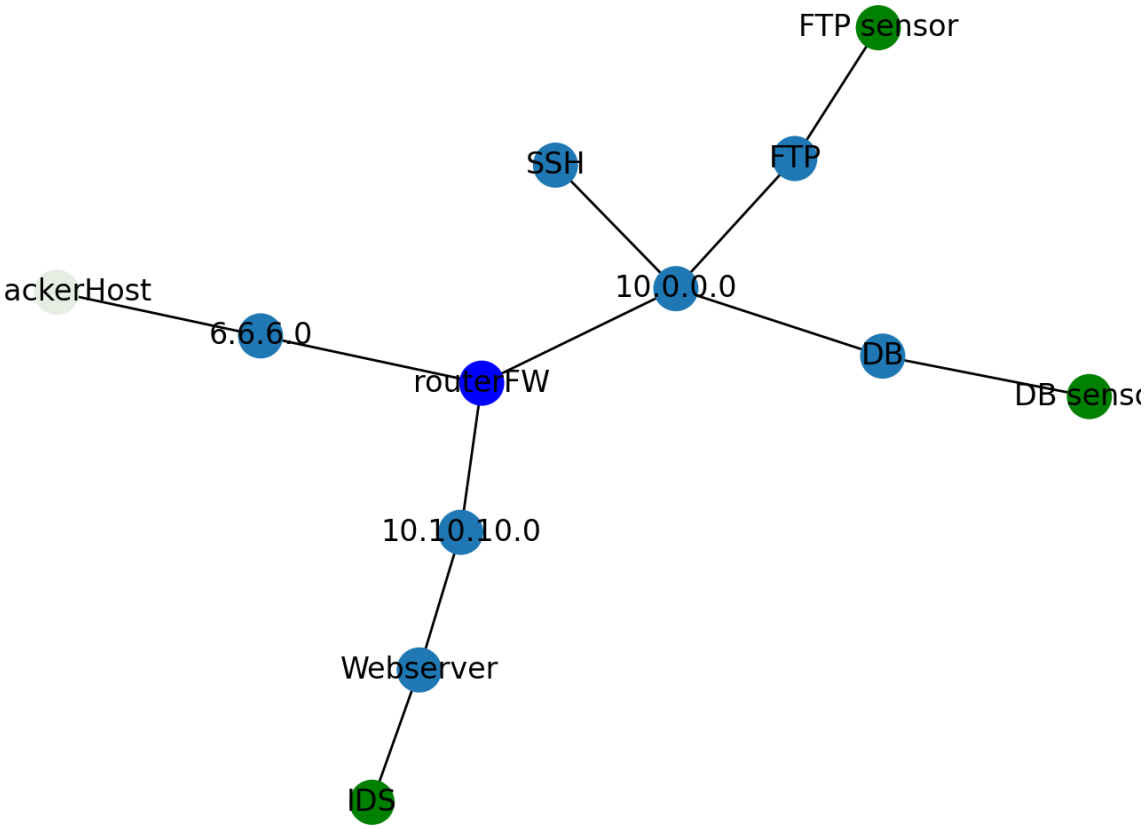

Figure 6. Simplified IMMUNE network model

3.1.2 IMMUNE Risk Assessment (IRA)

IMMUNE risk assessment model is responsible for computing the network risk. The underlying modelling technology is Bayesian networks, which is a graph-based representation for conditional probabilities. In the context of IMMUNE, Bayesian networks are useful to model the conditional probabilities of certain host being compromised with respect to the other hosts and sensor observation in the network, considering the attacker's possible attack paths. The IRA model allows to query probabilities that some host in the network has been compromised given the current sensor observation information. In the context of reinforcement learning, the IMMUNE Risk Assessment model will be used as defender's state observation.

3.1.3 Attack Graph (AG)

The attack graph is a model to represent the possible ways an attacker can perform lateral movement in the network. The attack graph is generated from the IMMUNE model by considering the vulnerabilities present in the network and the router's firewall rules to determine which parts of the network are reachable from different hosts.

The result is a directed acyclic graph with the attacker visibility vector as a vertex and vulnerability to be exploited to reach the next state as edge. The visibility vector represents attacker access privileges with respect to certain hosts in the network. The state values numbers have following meaning:

0. No visibility for attacker
1. Attacker has network access (can communicate, but some packets might be dropped due to network/firewall configuration) for the host
2. Attacker has access to the same subnet
3. Attacker has low privilege access on the host
4. Attacker has high privilege (root access) on the host

In the figure below, there is a resulting attack graph from the simplified IMMUNE network. There are 5 hosts in the network and the state vector represents the attacker's privileges in the following order: attacker host, Webserver, FTP, SSH, DB. For example, in the start of the attack, the state is [4,1,0,0,0] which means that attacker has root access on her host and has visibility on Webserver (since it is available on public web). By exploiting the vulnerability CVE-1 on the Webserver host, attacker obtains user level access on webserver, and now due to the configuration of the internal network, can also communicate with the FTP host, hence the new state value is [4,3,1,0,0].

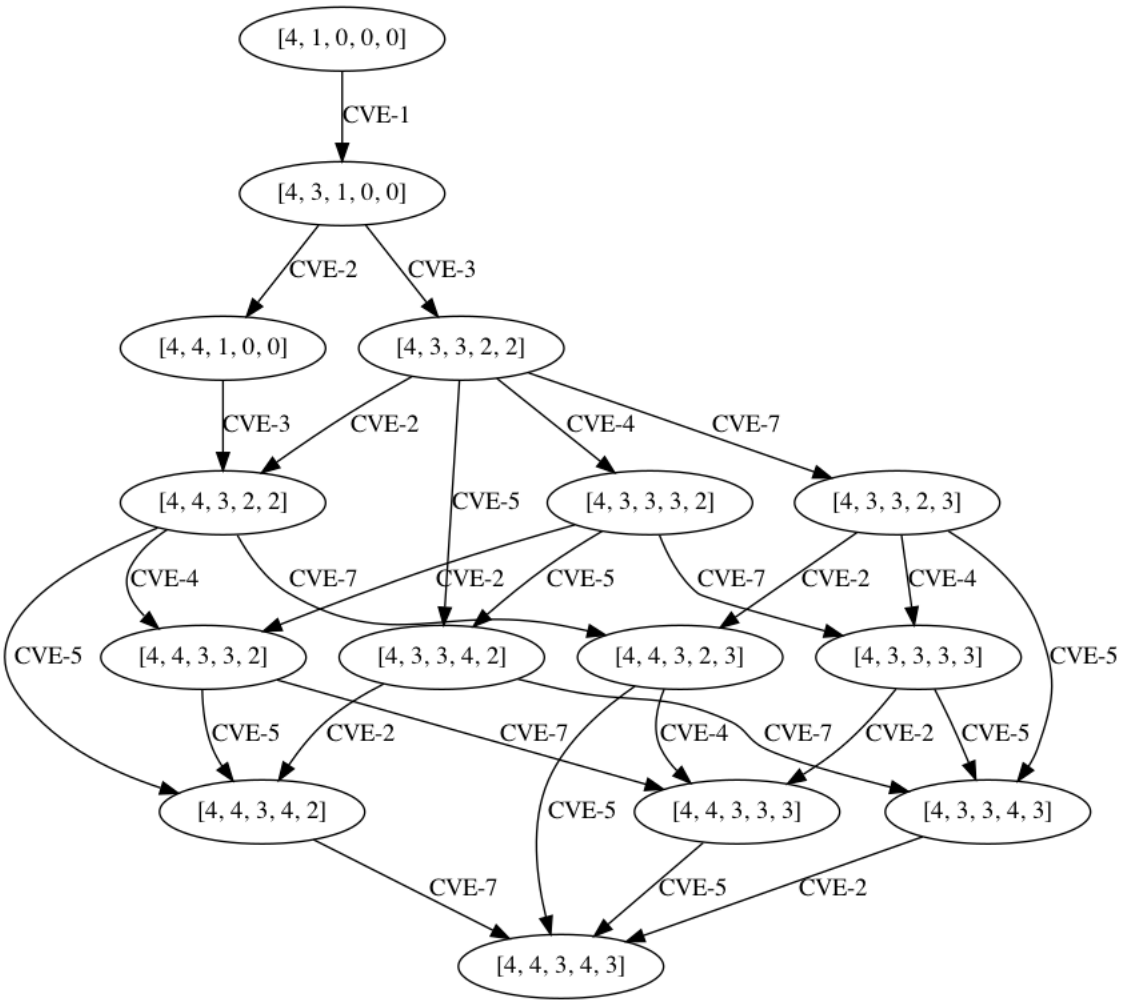

Figure 7. Attack graph of the simplified IMMUNE network

3.2 IMMUNE processes

The IMMUNE toolset has 3 different blocks of processes:

1. Network model construction
2. Risk analysis pipeline
3. Reinforcement learning cycle.

The figure below illustrates the processes and their interdependencies. The green colour illustrates the network model construction, grey colour illustrates the Risk analysis pipeline and the pink colour illustrates the Reinforcement learning cycle.

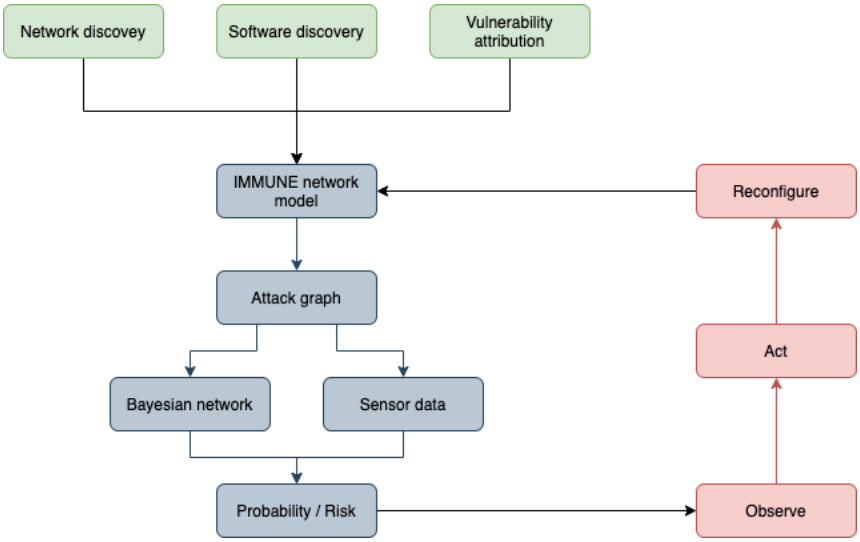

Figure 8. IMMUNE processes

3.2.1 Network model construction

The goal of network model construction is to turn the physical network with its switches, routers, interfaces and hosts into a representation of which it is possible to model the network risk and perform reinforcement learning. In other words, the goal of network model construction part of the pipeline is to turn the physical network into IMMUNE network model (introduced in section 3.1.1).

The main parts of network model construction part of the IMMUNE software are responsible for the following tasks:

- Network discovery
- Software discovery
- Vulnerability attribution

3.2.1.1 Network discovery

The network discovery is responsible for discovering the physical backbone of the network such as switches, routers and firewalls and active hosts in order to generate the IMMUNE network model. For the standard networks this step is performed using standard protocols such as SNMP, SSH and ARP. For software-defined industrial

networks, this step can be performed by querying the required information from the control plane.

3.2.1.2 Software discovery

Software discovery is the step to attribute the software that is installed on each component of the network. The standard that is used for this is called Common Platform Enumeration (CPE), which has following structure [16]:

cpe:/<part>:<vendor>:<product>:<version>:<update>:<edition>:<language>

The software discovery step is not possible to implement using common protocols, therefore the centralised database for each host's software needs to be created.

3.2.1.3 Vulnerability attribution

Having a centralised database of all the software used in a network in a structured and standardised way allows easy vulnerability attribution using openly available databases and the software CPE values. The source that IMMUNE project uses for this is CVE Search offered by Luxembourg's Computer Incident Response Centre [17].

3.2.2 Risk Analysis Pipeline and Reinforcement Learning Cycle

The goal of the IABG in the context of the IMMUNE project is to work on the network monitoring and risk assessment. In the risk analysis pipeline, the IMMUNE network model is used first to generate the Attack Graph representation. Together with the attack graph and the current network observation from the deployed sensors, the probabilities are derived for each node in the network indicating the likelihood that they are compromised by utilizing the Bayesian networks. The risk score for each component is computed by multiplying the derived probability with the component value score.

The main idea of the Reinforcement Cycle is for the defender to actively respond to the intrusions to the network by changing the configuration of the network in a way that minimizes the impact of the attack. The defender is trained by using adversarial learning where both attacker and defender are competing against each other in the same network. This process and methodology are explained in detail in chapter 4.

4 IMMUNE Reinforcement Learning

This chapter gives a more detailed overview, methodology and results of the implementation of the Reinforcement Learning Cycle that was introduced in the previous chapter.

This section describes the main contributions made to the IMMUNE project during the internship. The work started first with familiarizing with the concepts of reinforcement learning, the algorithms, and the concepts of software-defined networks presented in chapter 2. Next, the author got familiar with the current state of the project, on which a brief overview was given in the previous chapter. The goal after that was to design and implement the IMMUNE Reinforcement Learning Cycle and report the findings in order for the project team and the company to gain experience with the technology and assess the feasibility for the future projects.

4.1 Overview

The goal of using reinforcement learning in the IMMUNE project is to enable the defender to actively react to the intrusions reported by the sensors and the risk analysis pipeline in order to minimize the risk to its network. In order to train the defender to learn the optimal policy countermeasures for each stage of the attack, the attacker is simulated using the attack graph introduced in the previous chapter.

4.2 Previous Work with Reinforcement Learning in Cyber Defence

The researchers [18] from University of Groningen and TU Eindhoven have previously applied reinforcement learning in an adversarial cyber security setting. In the simulation, they modelled an environment, where the network has a set of nodes, each with a vector of values that represents attack and defence strengths that are only known to the attacker and defender respectively. During the attack, the attacker agent can change the attack strength while the defender can modify the defence values. Attacker's goal is to compromise the nodes in a network to reach to the final node which contains the asset Different learning algorithms were implemented such as Q-learning, deep learning, and various Monte Carlo algorithms such as UCB and simulated pairwise against each other.

The main findings from this simulation were that tabular and Monte Carlo algorithms performed much better than neural networks-based agents due to the fact that the latter ones were slower to adapt for the changes in the environment. They also discovered that while Q-learning was the best algorithm for the defender it was the one of the worst for the attacker side, which they believed is due to the slow convergence of the Q-learning algorithm.

4.3 Environment Setup

Both agents are operating in the same network environment with limited observability with respect to the other agent. The aim is to model the real-life scenario, where both the attacker and the defender might interfere their opponent's action, but not directly. The attacker must act with caution in order not to get caught, while the defender might notice some anomalous behaviour in the network, while not being able to pinpoint the exact source.

Both the defender and the attacker agents are operating in the same environment - the task can be considered as an adversarial setup between the attacker and the defender. More generally, the training loop looks as visualized in figure 9. In each step both attacker and defender receive their partial observation of the environment, and according to this knowledge they pick the best action that results with maximum reward (according to the algorithm they are using).

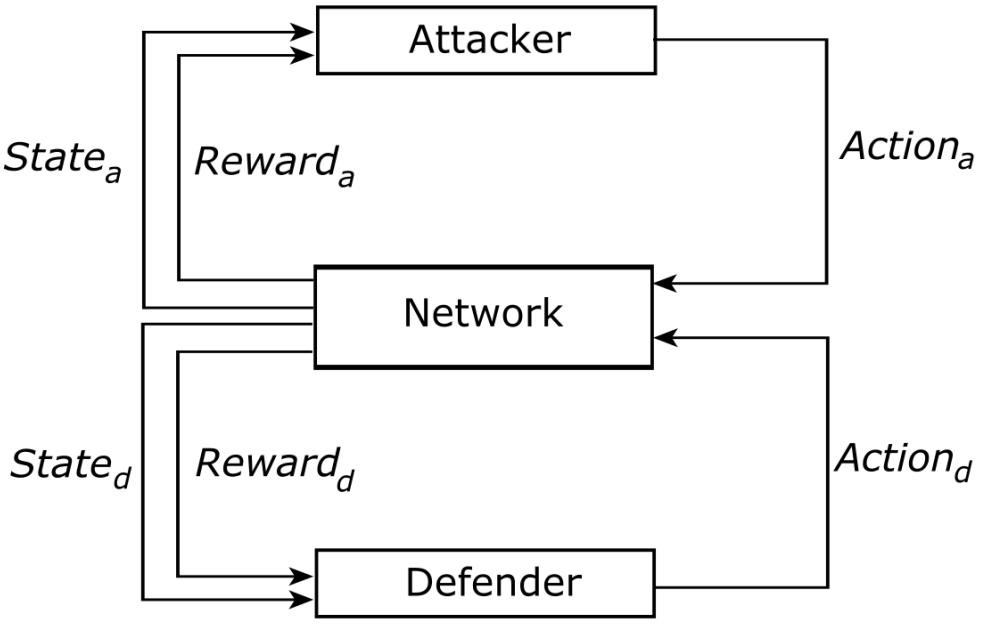

Figure 9. Agents Network Setup

Although, as mentioned previously, usually in multi-agent adversarial games, the reward function is a zero-sum, meaning that the reward is symmetrical between the winner and the loser. In IMMUNE's adversarial reinforcement learning, the strategy was selected to not use the symmetric reward. The rationale behind this decision is that the real-life cyber-attack scenario is asymmetric, as one side attacks and the other tries to defend. Usually, the defender does not have means to cause damage on the attacker but can only try to minimize the effect and damage while being attacked, and not win or gain any reward. Even if the attack is detected in the early phase, there is still costs and energy spent to deal with the incident from the defender side.

4.4 Attacker

The main idea behind an attacker is to simulate the intrusion by using the known vulnerabilities in the system. Attacker's environment is an attack graph (figure 7), which contains all the possible attack paths given the available vulnerabilities in the network, and also considering which hosts are reachable from each other with regards to the network configuration and the firewall rules. The attacker exploits different

vulnerabilities while moving laterally in the network. The attacker agent's state is its current position in an attack graph, which contains the information about the privileges and network visibility it has obtained for each known host in the network, so the attacker still has only partial observability with regards to the full network. Furthermore, as the defender takes countermeasures to mitigate the impact of the intrusion, the attack graph will change, so the environment for the attacker is dynamic. This will be described in more details later in the chapter.

The attacker's whole action space contains all the vulnerabilities available in the attack graph and during each timestep, the attacker can select between vulnerabilities available in its current position. Alternatively, the attacker can choose not to exploit any vulnerability and effectively stay in the same position. The state-actions space is presented in a table 2.

State	**Action**
• Position in an attack graph	• Do nothing • Select vulnerability v from all vulnerabilities

Table 2. Attacker State and Actions

For example, in the attack graph introduced in figure 7, the attacker's starting position is [4,1,0,0,0]. At this position, it can choose to exploit the vulnerability CVE-1 or alternatively, do nothing and stay in the current position. When an attacker decides to start an attack and exploit the vulnerability, its new position is [4,3,1,0,0], where in the next timestamp, the attacker can choose between doing nothing and exploiting either vulnerabilities CVE-2 or CVE-3. The agent receives positive reward for each of the vulnerabilities exploited and receives the cumulative result at the end of the episode.

To validate the attacker, the model 0 network configuration was used without the defender. First, when the attacker was run with a function where it receives fixed positive reward for every vulnerability exploited, the attacker intuitively learned to take the longest path between the start and terminal state. When the reward function was modified

as the attacker would receive negative reward for each of the vulnerability exploited, on the contrary to the previous experiment, the attacker now learned to take the shortest path between start and terminal state.

4.5 Defender

The goal of the defender agent is to learn the optimal policy of countermeasures to react to the attack effectively. The defender operates on the network model, which it has full knowledge of, however, the defender cannot observe the attacker directly and has to rely on the signals coming from the sensors which are stochastic in their nature, which means that there is a small probability of false positive and false negative readings.

The defender's state consists of sensor readings that have been processed by the risk assessment system (from the risk analysis pipeline) and returns the probability for each host in the network whether it is compromised by the attacker given the current network observation from the sensors.

As IMMUNE uses software-defined networking as its underlying technology, it is easy to reconfigure the network centrally via the control plane. This idea is captured in the defender's action space, where it has a set of possible configurations of the network and based on the observation it can choose either to stick to the given network configuration or change to a different one.

State	**Action**
• Network observation – P(host = compromised | observation) for each host in the network • Index of current configuration c from all configurations	• Do nothing • Select new configuration c' from all configurations

Table 3. Defender state and actions

In reinforcement learning, the agent's goal is to learn a policy that maximizes the reward function. The formulated goal for the internship is for the defender to learn

countermeasures that minimize the risk to the system, in this case the total risk to the system is computed by summing up the risk scores of individual hosts. The goal of the defender is to also minimize the impact of the attack, so the duration of the attack needs to be considered as well. Finally, as the goal is to minimize the number of configuration changes, the agent receives a penalty for each network reconfiguration.

Taking all of these requirements into account, the defender's reward function has 3 components:

1. Costs of action taken R_a – fixed penalty for each configuration change
2. Duration of the attack R_t – fixed penalty for each time step of attack duration
3. Cumulative risk R_r – Risk score for each host during each step of the simulation.

As similar to attacker, the agent receives the cumulative reward in the end of each episode that is sum of the 3 components:

$$R = R_a + R_r + R_t$$

Note here that, as opposed to the attacker, the defender's reward is negative for all of the components, so in fact, while the agent is maximizing its reward, it is actually minimizing the negative reward.

4.6 Experiments Setup

The experiments are run on the simplified IMMUNE network (figure 6) with 5 different hosts: Attacker, Webserver, SSH, FTP and DB and some sensors to detect anomalies. The network firewall is configured in a way that the Webserver is in a demilitarized zone (DMZ), but due to misconfiguration it can communicate with the FTP host. There are also 7 different vulnerabilities present in the network model, which enable attacker for lateral movement in a network, that is represented in the resulting attack graph (figure 7).

As mentioned in the previous paragraph, the defender has a set of configurations it can apply in order to mitigate the attack. The underlying software-defined networking technology used in IMMUNE network allows for fast and centralized

For experiment purposes, there were 3 additional configurations given to the defender to apply. In the default network the SSH, FTP, and DB hosts are all located in the same subnet 10.0.0.0/24. In each of the reconfigurations, there is one host that is 'moved' to the different subnet 10.0.1.0/24. Since the 'moved' host is now in a different subnet, and cannot communicate with any other host, the attacker might not be able to use all the possible vulnerabilities for lateral movement.

The first configuration (noted as model0) is the simple network model (figure 6), as already mentioned, the model contains several vulnerabilities which can be used for lateral movement by the attacker. All the possible attack paths are illustrated in the attack graph (figure 7).

In the second configuration (noted as model0) of the network (figure 10), the FTP host is in a subnet 10.0.1,0/24.

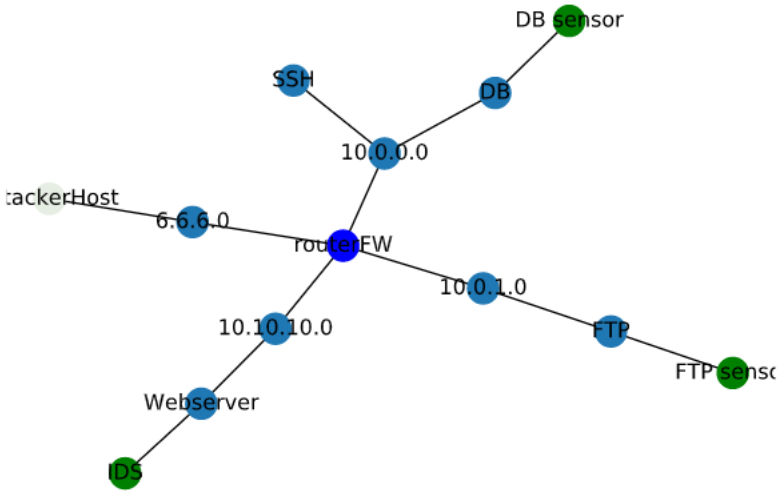

Figure 10. Network configuration model1

In model0, the network was configured in a way that the Webserver and the FTP host were able to communicate with each other, making it possible to exploit vulnerabilities in an FTP host and after that compromise other hosts in 10.0.0.0/24 subnet. In model1 the FTP host is in the 10.0.1.0/24 subnet, which is isolated from other network segments. Meaning that the attacker can still attack the Webserver and gain root privileges there,

but the rest of the lateral movement is not possible. The resulting attack graph is presented in a figure 11.

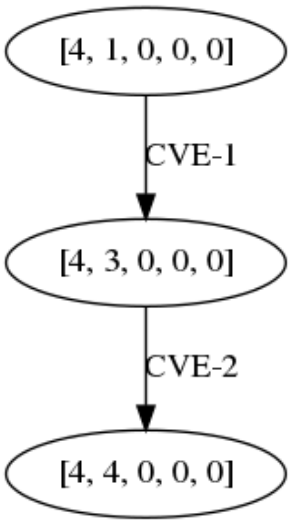

Figure 11. Attack graph model1

In the third configuration (noted as model2) of the network the SSH host is in the 10.0.1.0/24 subnet and in the fourth configuration (noted as model3) of the network the DB host is in a separate subnet. In both cases, the vulnerabilities from either SSH or DB host are not exploitable for the attacker, meaning that some of the vulnerabilities are not represented in the attack graphs (figures 12 and 13).

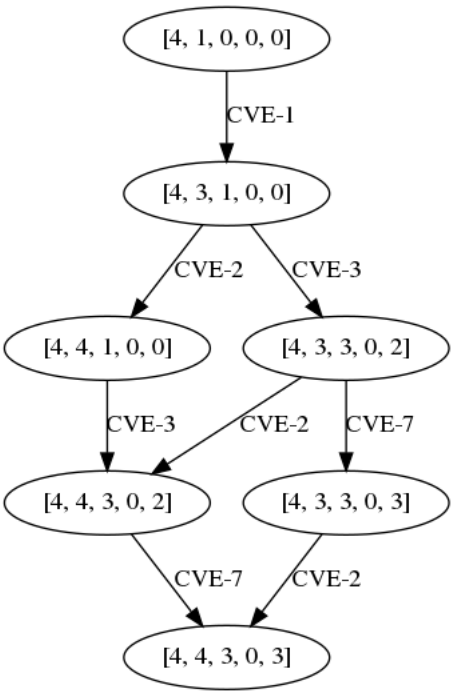

Figure 12. Attack graph model2

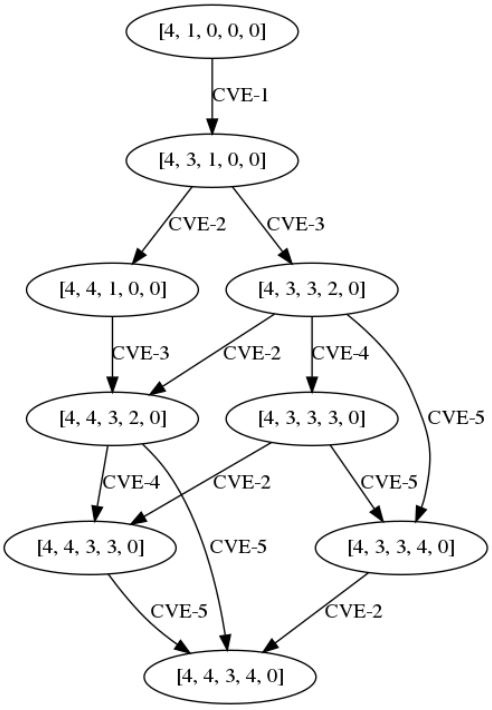

Figure 13. Attack graph model3

In the experiments, the defender can switch between those four configurations resulting in changes in the attack graph, and therefore the attacker's environment is constantly changing as well. In the conducted experiments, the starting point is model0. Agents using different algorithms were implemented in both attacker's and defender's side. Implemented algorithms include: Q-learning, Deep Q-learning, Deep-Q learning with experience replay and A2C. In addition to the reinforcement learning agents, a random agent was also implemented that takes random action during each step. During the experiments 1000 episodes were used and all implemented agents were simulated against each other. The metrics used to compare the agents are steps, attacker and defender rewards averaged for each episode.

4.7 Experiments Results

The goal of the experiments was first to run attacker and defender against each other in an adversarial simulation starting with Q-learning and validate that the agents are able to learn. Following the validation, the results are interpreted by visually interpreting the agent's behaviour after the training process. Finally, all implemented agents were simulated against each other.

4.7.1 Proof of Learning

First learning validation run was conducted using Q-learning agents for both attacker and defender sides for 1000 episodes. The metrics kept track on were steps per each episode and the attacker and the defender rewards. The average for these metrics during the initial training run were:

- **Mean steps per episode** – 5.04
- **Mean attacker reward per episode** – 33.05
- **Mean defender reward per episode** – -115.31

In figure 14 is the plot of the average steps per each 10 consecutive episodes. The plot illustrates how the steps converge already after ca 50 episodes. Same behaviour can be observed with both attacker and defender rewards (figure 15) where similar convergence happens.

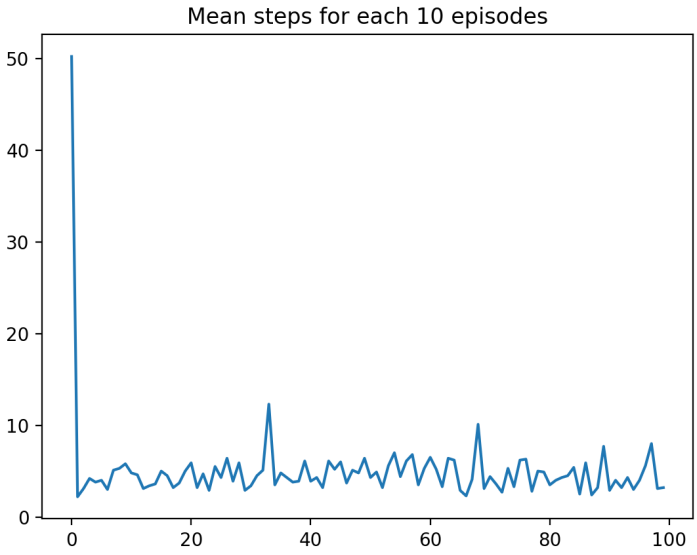

Figure 14. Q-learning steps

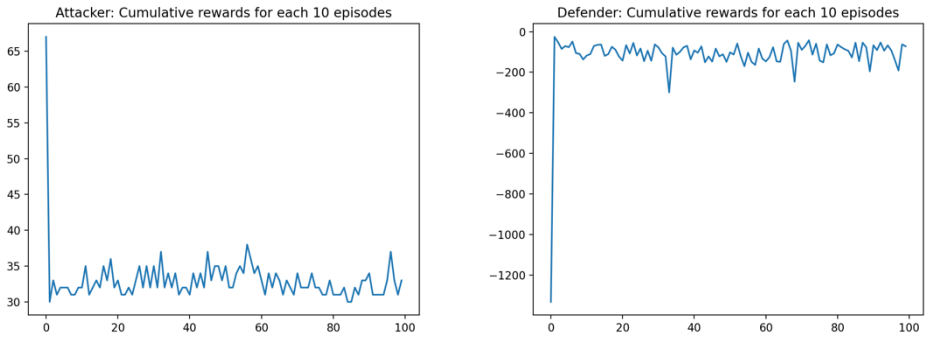

Figure 15. Q-learning rewards

4.7.2 Learned Policy Interpretation

Good way to interpret the attacker's and the defender's learned behaviours is to analyse the actions that the trained agents take. In order to do so, the attack graph is used to visualize the attacker's movement and current configuration. The trained Q-learning agents from the previous paragraph are used and a single episode is played with agent's only taking the best actions according to their learned policy.

4.7.2.1 Step 0

The start of the episode (figure 16), the attacker is in a position [4, 1, 0, 0, 0] and the network uses the first configuration (model0).

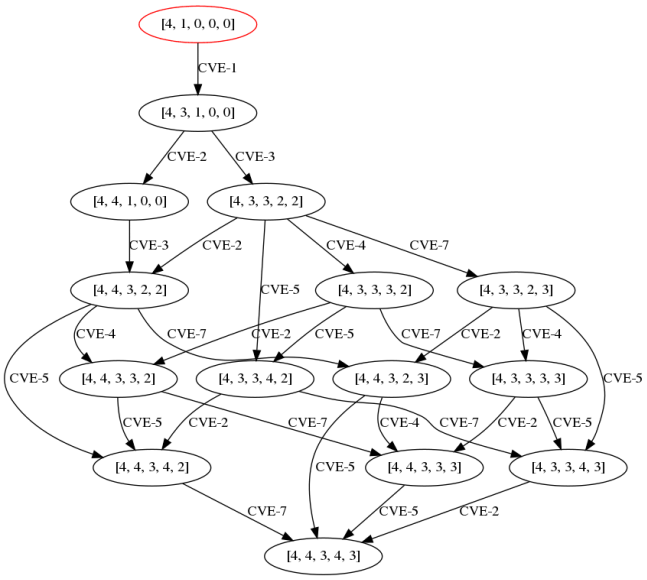

Figure 16. Scenario - Step 0

4.7.2.2 Step 1

Attacker starts the lateral movement by exploiting CVE-1 on the Webserver host and gains the user-level privileges there. New attacker's state is [4, 3, 1, 0, 0].

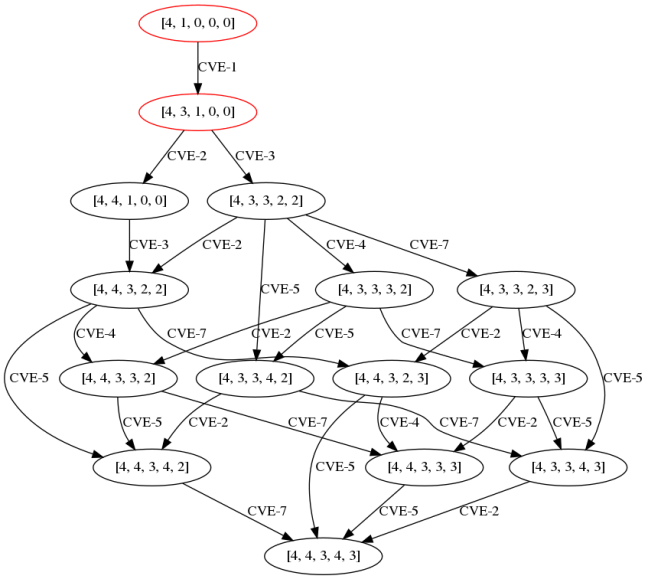

Figure 17. Scenario - Step 1

4.7.2.3 Step 2

Defender reacts to the attack by changing the network configuration to model3 (figure 18). The attacker remains in the same state and still has the opportunity to exploit either the CVE-2 or CVE-3 next.

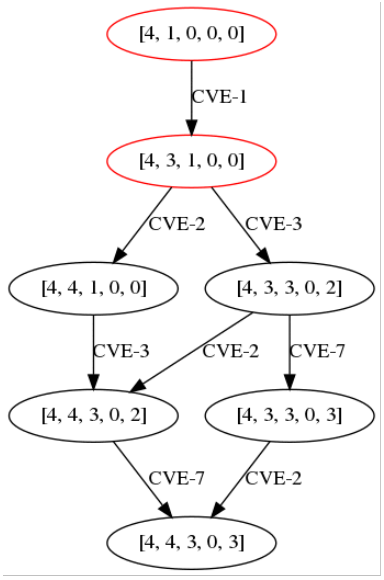

Figure 18. Scenario - Step 2

4.7.2.4 Step 3

The attacker continues by exploiting the CVE-2, which gains her root privileges on Webserver (figure 19). Her new state is [4, 4, 1, 0, 0].

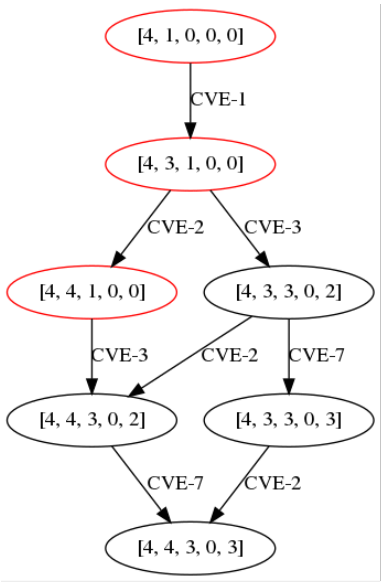

Figure 19. Scenario - Step 3

4.7.2.5 Step 4

Defender switches to configuration model2 and since the attacker is in the final state of the attack graph the episode finishes as the terminal state has been reached.

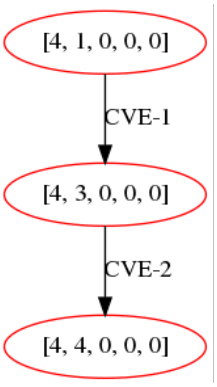

Figure 20. Scenario - Step 4

4.7.2.6 Interpretation with Risk

Defender reward components consist of penalty for time of the attack, number of configuration changes, and the total risk. As the goal of the reinforcement learning agent is to maximize the long-term rewards (in the case of defender to minimize the negative reward) the agent must try to make sure that the episode is as short as possible, with minimum amount of configuration changes and lowest total reward.

In figure 21 below, there is a plot of the risk score from the risk assessment system in the y-axis for each of the configurations with different sensors being activated in the x-axis. Intuitively, the total risk scores match with the size of the attack graphs and attack surfaces for different network models, and also larger the number of intrusion detection sensors being activated correlates with higher risk. Therefore, in order for the defender to minimize the risk and duration of the attack, the defender should switch to a configuration with lower risk and least number of vulnerabilities that result with less steps during the attack.

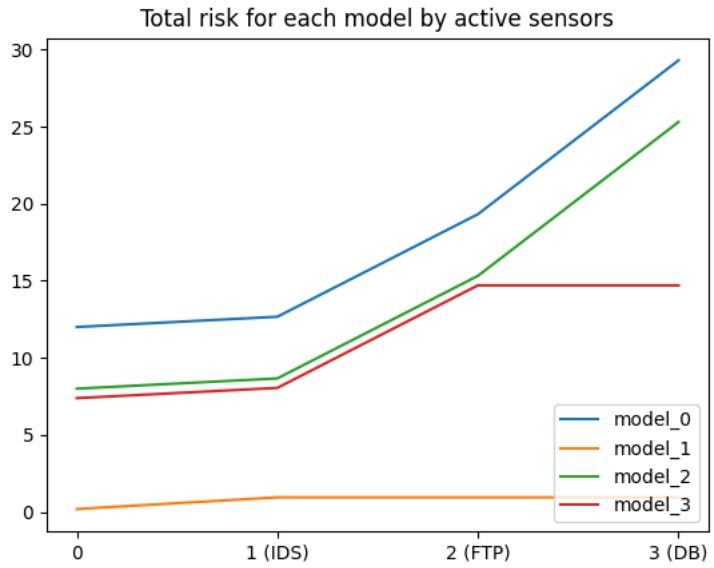

Figure 21. Configurations' risk

4.7.3 Simulation

In order to analyse how well different algorithms perform on both attacker and defender side, the experiment was conducted where all different implemented algorithms are run against each other and the average steps, attacker and defender rewards are measured. The simulation is 1000 episodes long and to minimize one-off statistical anomalies for each attacker and defender pair, the simulation was repeated 10 times, and the average values from there were used for the comparison.

4.7.3.1 Steps

From the defender's side, the biggest outlier is A2C defender with much higher average steps than other agents. Other attacker-defender pairs seem to perform similarly with the lowest average number of steps from both attacker's and defender's side is Q-learning.

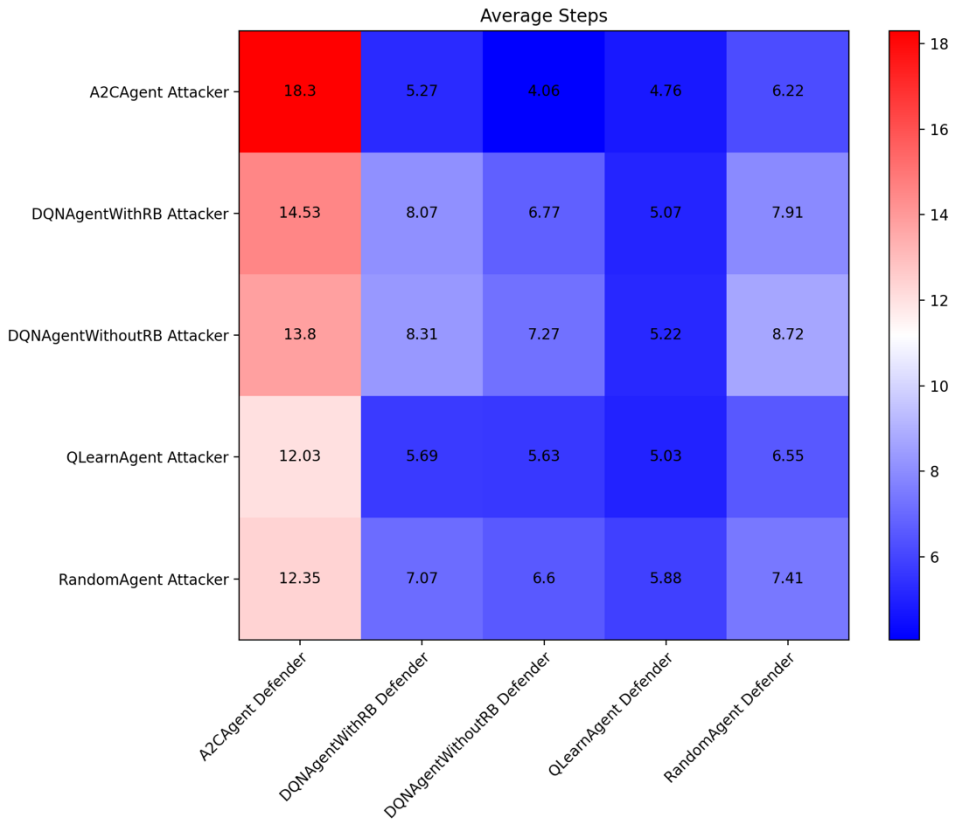

Figure 22. Steps Heatmap

4.7.3.2 Defender's Reward

When analysing the defender's reward, the first thing to notice is that similarly to the number of steps, the A2C agent performs the worst on both the attacker and the defender side. On defence it has the highest number of negative reward and during the attack on average all the defenders have managed to gain the highest amount of reward against it. The best overall defender is the Q-learning agent with the highest amount of reward against all attackers.

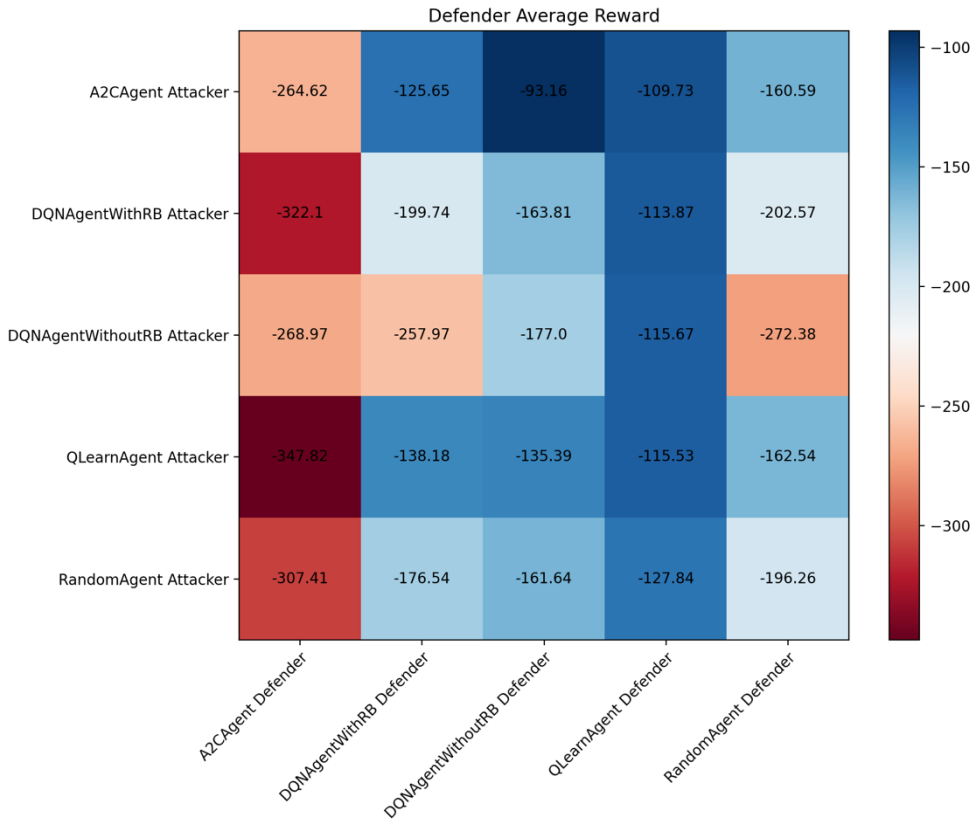

Figure 23. Defender Reward Heatmap

4.7.3.3 Attacker's Reward

When analysing the reward from the attacker's perspective it also stands out that all agents have received on average the most reward against the A2C defender agent. The best performing attackers are Deep Q-learning agents. From the defender's side, again all attackers have managed to gain on average the least amount of reward against the Q-learning agent.

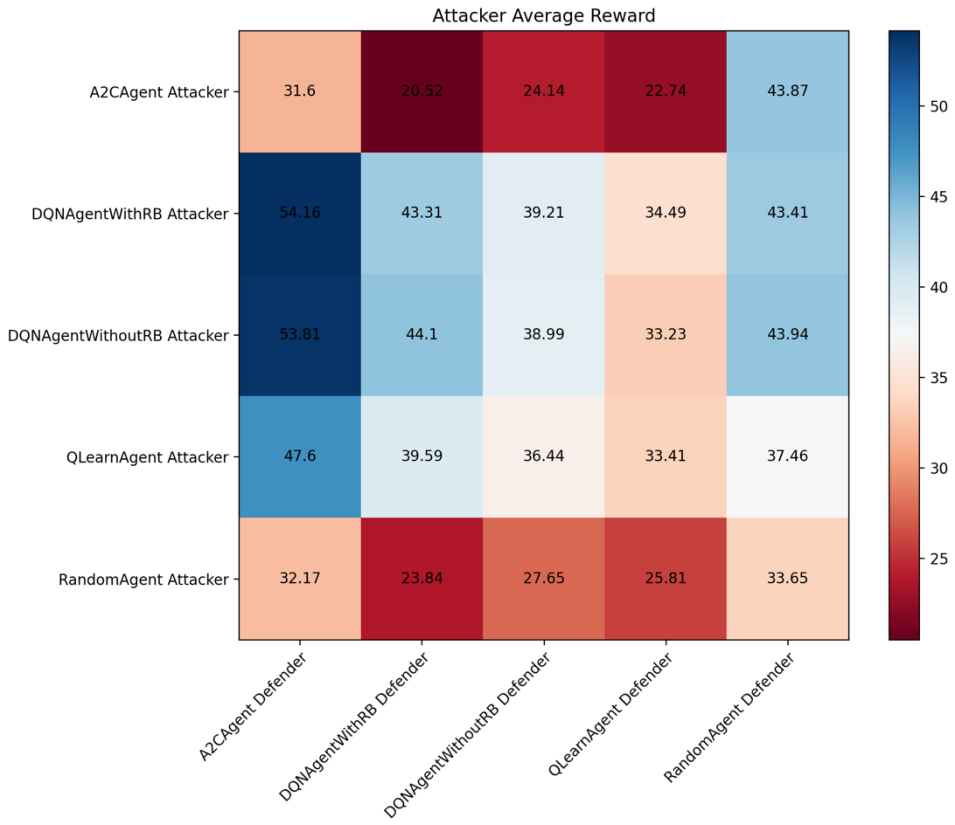

Figure 24. Attacker Reward Heatmap

4.8 Discussion of Results

The goal of simulating the attacker and defender against each other was to validate whether reinforcement learning approach could be used to simulate the attacker in order for the defender to learn countermeasures. With this proof of concept, it was validated that indeed the defender can learn from the risk scores to change to network configuration with lower risk.

Similar to the results from [18], the overall best defender agent is Q-learning while during the attack the Deep Q-learning agents outperformed it. Based on the conducted experiments, the Advantage Actor Critic algorithm does not seem to be the best solution for a dynamic environment. One of the reasons for poor performance could be that it has been discussed in literature [19] [20] that policy gradient methods can be highly unstable, and since the adversarial multi-agent environment is highly dynamic and non-stationary, this combination could lead to bad performance.

4.9 Further Work and Possible Improvements

The limitations of using attack graphs generated by known public vulnerabilities means that although it would be possible to accurately simulate the attacker for known vulnerabilities, it also means that the same is not possible for not yet known or publicised vulnerabilities. This is a downside since industrial networks could be targeted by highly sophisticated attackers. For example, the Stuxnet virus launched against Iranian nuclear facilities that relied on similar PLCs used four different zero-day vulnerabilities [21]. Furthermore, as the attack graphs rely on publicised CVEs, they are not covering cases where cyber-attacks start from phishing emails, which trick the employees to install malware to their systems and by doing that give attackers persistence, these kind of attacks do not exploit any technical vulnerabilities and rather rely on social engineering.

Possible improvement of the IMMUNE reinforcement learning approach, that IABG is already working on, is how to include the cost model to different configurations in the network in a way it could be also used as defender's reward function. The solution developed in the context of this proof of concept does not consider the possibility that some reconfigurations might be really expensive due to their impact on the business process.

Possibly the biggest prerequisite for using a reinforcement learning approach for network defence is having good sensor observations, which was not yet validated on a real-world operational network or with network data due to fact that the state of the IMMUNE project is not there yet.

4.10 Knowledge Transfer and Project Handover

As reinforcement learning is a relatively new technique for the department and the company, an integral part of the internship was to ensure that the knowledge would stay in the IMMUNE team even after the end of the internship. Furthermore, as IMMUNE consists of many interdependent software packages and components (introduced in chapter 3), the code implementing the reinforcement learning functionality was developed from the beginning with an aim to make use of the other components and become itself as a part of the bigger IMMUNE software framework.

This means that the software repository written during this internship must remain maintainable even after the internship. In order to do so, following strategies were used:

- Documenting the code and methods
- Unit testing
- Code handover sessions with IMMUNE team

First, an extensive documentation was created that included documentation of every class and function, usage guide, and a documentation in the format of an internal website about the background of reinforcement learning and experiments, which is essentially a more condensed version of this report. Second strategy to make sure that the software is reliable and easier to make further changes to, was to cover the code with unit tests. The final unit test line code coverage was 87%. Third, to make sure that the team members have sufficient understanding on how the software works, multiple handover sessions were conducted, where the whole codebase was covered and explained.

5 Conclusions

The goal of this work was to implement and then analyse the feasibility of the reinforcement learning in an autonomous network self-defence with underlying technology as software-defined networking. As a result of this work, the adversarial simulation was developed which utilises attack graphs to simulate attacker and the IMMUNE risk assessment system for defender to learn the best countermeasures to defend against an attack. Finally, several different reinforcement learning algorithms were implemented and compared against each other.

As a result, a proof-of-concept simulation the defender was able to learn to reconfigure the network in a way that results with lower risk and shortest attack paths. However, there is still further work to be done on the intrusion detection, cost modelling and finally deployment of the system. Among the different algorithms that were implemented, the Q-learning performed best on the defender side and deep Q-learning from the attacker side.

During the latter stage of the internship, the main focus was on knowledge transfer on reinforcement learning and documenting my experiment code as the IABG is also looking to expand on this work, and also find other applications for reinforcement learning in other projects. There is already a proposal of another project which relies heavily on the developed reinforcement learning framework from IMMUNE, with an extension for different types of attackers and different network types

The main contributions on this internship were to show that it is possible to train the defender to deploy countermeasures to the attacks based only from the output of the risk assessment system. This work can be used as a framework where the attacker is simulated using the specific vulnerabilities and attack vectors of the given system to train the defending agent. Furthermore, the approach can also generalise to different types of countermeasures as the actions in reinforcement learning is an abstract concept, so the underlying system does not need to be implemented by relying on software-defined networking concepts, but can define its own countermeasures which the reinforcement learning agent can use as actions.

It must be also considered that developed approach has some drawback and possible future improvements. First, it relies on the of network sensor information, which may not be always possible to obtain or may contain noise. Furthermore, this approach does not consider zero-day vulnerabilities or social engineering attacks such as phishing emails.

The main lessons that can be learned from this work is that competitive multi-agent setup can be used as a tool to simulate the cyber-attacks and the response. This setup can be useful in cases in where real datasets are not available or easily accessible. Furthermore, based on the goal of the research, this setup can generalize to different types of attacker and defenders, which are modelled according to their respective goals.

References

[1] [Online]. Available: https://www.iabg.de/en/. [Accessed 1 June 2020].

[2] "Industrial members," [Online]. Available: https://www.eurecom.fr/en/partners/partnership-policy/industrial-members. [Accessed 1 June 2020].

[3] "Projektübersicht," [Online]. Available: https://immune-projekt.de/about/. [Accessed 1 June 2020].

[4] "IMMUNE," [Online]. Available: https://www.ifak.eu/en/projects/immune-self-defence-networks-resilient-industry-40. [Accessed 1 June 2020].

[5] R. S. Sutton and G. A. Barto, Reinforcement Learning: An Introduction, 2018.

[6] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Machine Learning Proceedings*, Brunswick, 1994.

[7] C. Watkins and P. Dayan, "Q-Learning," *Machine Learning,* no. 8, pp. 272-292, 1992.

[8] V. Mnih and K. Kavukcuoglu, "Playing Atari with Deep Reinforcement Learning," 2013.

[9] V. Mnih and A. Badia, "Asynchronous Methods for Deep Reinforcement Learning," *arXiv:1602.01783v2,* 2016.

[10] M. Lee, "Actor-Critic Methods," [Online]. Available: http://incompleteideas.net/book/first/ebook/node66.html. [Accessed 12 July 2020].

[11] G. Dulac-Arnold and D. Mankowitz, "Challenges of Real-World Reinforcement Learning," *arXiv,* 2019.

[12] K. Zhang and Z. Yang, "Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms," 2019.

[13] K. Benzekki and A. El Fergougui, "Software-defined networking (SDN): a survey," *Security and Communication Networks,* vol. 9, no. 18, pp. 5803-5833, 2016.

[14] D. Kreutz and F. Ramos, "Software-Defined Networking: A Comprehensive Survey," *arXiv,* 2014.

[15] [Online]. Available: https://nvd.nist.gov/products/cpe. [Accessed 29 June 2020].

[16] "Common Platform Enumeration (CPE)," [Online]. Available: https://nmap.org/book/output-formats-cpe.html. [Accessed 3 July 2020].

[17] "CVE search," [Online]. Available: https://www.circl.lu/services/cve-search/. [Accessed 3 July 2020].

[18] R. Elderman and L. Pater, "Adversarial Reinforcement Learning in a Cyber Security Simulation," *Proceedings of the 9th International Conference on Agents and Artificial Intelligence,* 2017.

[19] R. Houthooft and R. Chen, "Evolved Policy Gradients," 2018.

[20] T. Zhao and H. Hachiya, "Analysis and Improvement of Policy Gradient Estimation," *Neural Networks,* vol. 26, pp. 118-129 , 2012.

[21] R. Naraine, “Stuxnet attackers used 4 Windows zero-day exploits,” 14 September 2010. [Online]. Available: https://www.zdnet.com/article/stuxnet-attackers-used-4-windows-zero-day-exploits/. [Accessed 13 August 2020].