AALBORG UNIVERSITY

DENMARK

Master's Thesis
Mathematical Engineering

Malware Detection utilizing Reinforcement and Federated Learning

Author:
Martin Møller Sørensen

Supervisors:
Shashi Raj Pandey
Christophe Biscio

June 3, 2024

Dept. of Mathematical Sciences
Skjernvej 4A, DK-9220 Aalborg Øst
http://math.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Malware Detection utilizing Reinforcement and Federated Learning

Project Period:
February 2024 - June 2024

Participant:
Martin Møller Sørensen

Supervisors:
Shashi Raj Pandey
Christophe Biscio

Pages: 32

Date of Completion:
June 3, 2024

Abstract:

In the rapidly evolving landscape of cybersecurity, malware detection provides a complex and dynamic challenge. This thesis explores innovative techniques in the reinforcement learning and the federated learning domain to enhance the robustness and sophistication of malware detection systems. This thesis has developed a reinforcement learning framework, which incorporates components of the Rainbow reinforcement learning approach, there has demonstrated superior results in Atari games and similar potential advancements could be achieved in the realm of malware detection. To quantifying the results of the proposed reinforcement learning framework, a baseline utilizing the conventional deep Q-network was established. The purposed framework was evaluated in a centralized and decentralized environment.

Preface

This thesis concludes my final semester of the Master program Mathematical Engineering at the Department of Mathematical Sciences at Aalborg University. This thesis was written in the time period between February 2024 and May 2024.

For the thesis, it is preferred that the target audience is sufficiently familiar with the terminology inherent to the domain of machine learning and reinforcement learning. Additionally, terminology specific to the domain of cybersecurity is employed for further information.

The script related to this thesis was written in `Python` 3.11, and is accessible in the attached folder, `MAT-TEK10`.

The author thanks Shashi Raj Pandey (Department of Electronic Systems) and Christophe Biscio (Department of Mathematical Sciences) for the support and guidance throughout the project's writing process.

Aalborg Universitet, June 3, 2024.

Nomenclature

Acronyms

Description	Acronym
action, next action and action space	$a, a', \mathcal{A}$
state, next state and state space	$s, s', \mathcal{S}$
reward and return	r, G
value funtion and action-value function	V, Q
discount factor and learning rate	γ, α

Abbreviations

Description	Abbreviation
Deep Q-Network	DQN
Double Deep Q-Network	DDQN
Federated Learning	FL
Importance-sampling	IS
Machine Learning	ML
Mean Squared Error	MSE
Neural Network	NN
Prioritized Experience Replay	PER
Reinforcement Learning	RL
Denial of Service	DoS
Intrusion Detection Systems	IDS
Remote to Local	R2L
User to Root	U2R

Contents

1 | Introduction

In the rapidly evolving landscape of cybersecurity, malware detection provides a complex and dynamic challenge, which can be compared to a strategic game between two players, similar to chess. One of the players represent the attacker, the cybercriminals, which is both the creator and distributor of malicious actions. The other player corresponds to the defender, cybersecurity analysts or detection systems, where the objective is safeguarding the modern digital and interconnected landscape, as advancements in technology rapidly advances. The game occurs in the digital world, dominated by creative thinking, where both the attacker and the defender are regularly reinventing existing strategies to obtain an advantage.

The game of malware detection is centralized around keywords such as identification and mitigation of malicious actions. The attacker has an information asymmetry, which provides the cybercriminals with an advantage, as they determine when and where to launch an attack. Firstly, a reconnaissance of the target victim is performed, to identify vulnerabilities that potentially can be exploited. Cybercriminals have designed malicious software with the purpose of infiltrating and compromising victim networks or devices. This provides a significant threat against not only the individuals, but also businesses and governments worldwide.

The defender fall short in options as it is infeasible to employ all existing malware detection approaches, and further individual strategies results in various outcomes. This restricts the defender in the decision of strategy, as overlaps in capabilities of approaches result in a wasteful utilization of the available resources. The primary objective for the defender is to develop sophisticated detection systems, that have the capacity to accurately and consistently identify and mitigate malware threats before the attacker have the opportunity to inflict any damage.

1.1 Motivation

With the tremendous influence of the digital world on our physical realm, the technological advancement of the physical world progresses in the direction of industrial 4.0, which introduces new levels of intelligence capabilities. The never-ending rapid growth of data is a response to the technological advancement and interconnected landscape, introducing an unprecedented challenge, the threats of cyberattacks being more critical and rapidly advancing [14, p. 1]. Cyberattacks are quite expensive, and a successful data breach could potentially result in a devastating impact to the entire nation, during the potential consequences of unauthorized access or manipulation of data. The devastating impact of a data breach of sensitive data, spans from financial losses to reputational damage, and can be permanent. As organizations store vast amount of data, the requirement for robust and sophisticated cybersecurity mitigation strategies becomes paramount.

The cybercriminals continually refine their techniques to circumvent detection approaches, and continue to pose the threat of exploiting vulnerabilities in systems and networks. The threat spans from ransomware attacks targeting critical infrastructure to sophisticated phishing attempts to steal sensitive information. The cybercriminals launch approximately 2200 cyberattacks every day on a global scale [14, p. 1]. In fact, zero-day attacks remain a

significant challenge to the cybersecurity domain. This category of attacks includes novel and unseen malware, where conventional cybersecurity approaches fall short. Reinforcement Learning (RL) exhibits the capabilities to learn from previous experiences and adapt to the complex and dynamic attack surface.

In cybersecurity, universally representative datasets of the real world are critically important for developing reliable RL agents. The drawback of centralizing sensitive data, is to maintain data privacy and confidentiality. Federated Learning (FL) mitigates these challenges, and enables collaborative model training across decentralized environments, without the requirement of data centralization. An advantage of utilizing a decentralization approach is reducing the risk of data exposure and preventing the sharing of sensitive information.

1.2 Purpose and Objective

This thesis will contribute to the research field of malware detection by integrating the Rainbow RL approach in combination with FL. The primary objective is to develop a reinvent strategy for malware detection that prioritizes data privacy preservation.

The purpose of this thesis is to explore innovative techniques in both the RL and FL frameworks, and their application to malware detection systems. It is anticipated that the results achieved in this thesis will serve as a factor in the consideration of the integration of this innovative thinking of the combination between enhanced RL approaches and FL framework, into the current infrastructures. As threats and vulnerabilities can arise in any technology, the research of innovative cybersecurity approaches is of paramount importance to guarantee the security of our interconnected digital global network.

The objective of this project is to develop a RL-based framework for malware detection, with the purpose of exploring its potential within this domain. The Rainbow RL approach has demonstrated superior results in Atari games [6, p. 2], and similar potential advancements can be achieved in the realm of malware detection. The primary focus is to compare componentes of this advanced algorithm with the widely utilized conventional Deep Q-Network (DQN) algorithm. The DQN RL approach will serve as a central baseline, while the components of the Rainbow RL approach will be examined both in a centralized and decentralized context. This study will apply the well-established dataset NSL-KDD, which contains various types of attacks divided into distinct classes. This dataset will provide a reference point for evaluating the performance of the proposed RL-based framework for malware detection.

1.3 Outline

The Rainbow RL approach will probably be a novel concept for the reader, therefore Chapter 2 will present an overview of the variety of enhancements related to the DQN RL algorithm, which will contribute to the Rainbow RL framework. Chapter 3 will outline FL, where a comparison between centralized and decentralized approaches is presented. In Chapter 4, details about the system setup, including various configurations, is described and an overview of the dataset that has been utilized in this study will be presented. Chapter 5 examines the results for various scenarios, reflecting the effectiveness of the proposed RL-based framework. Finally, Chapter 6 will conclude the thesis, and discuss the possibility for potential enhancements

2 | Deep Q-Network

This chapter will attempt to provide an overview and introduce the terms relating to the field of DQN RL, and to discuss the aspects of enhancements that collaborate to the Rainbow RL framework. This chapter is primarily based on [13, pp. 3-7, 10-12] and [6, pp. 1-4].

2.1 Prerequisites

RL is a subfield of Machine Learning (ML), where the purpose is to enable an agent to acquire information through interacting with an environment. Initially, the agent lacks explicit knowledge about the decisions, but through trial and error, the agent discovers which decisions lead to the most desired outcomes. Below, some major aspects associated with RL are outlined.

- **Agent:** The agent is the system decision-maker that interacts with the environment by performing a variety of actions, making observations, and achieving rewards or penalties in return. In practical terms, the agent is the software that attempts to solve a virtual representation of a real world challenge.
- **Environment:** The environment is the external system with which the agent interacts. The communication between the environment and the agent is limited to the following terms, observations, actions, and rewards. The environment can be either a physical system or a virtual world.
- **Observations:** Observations are the information that is perceived from the environment. The agent will receive an observation/state, which is representing the agent vision that captures a snapshot of the current configuration of the environment, required for the decision-making.
- **Actions:** Actions are the agent's abilities to interact and alter the environment. The set of actions corresponds to all the feasible decisions that the agent can perform in a particular state. An action can either be continuous or discrete.
- **Rewards:** Rewards are numerical values that are periodically provided by the environment as feedback to the agent. A reward is a number that can be positive or negative and large or small. The objective of the agent at time-step t is to maximize the expected accumulative reward, defined as:

$$G = \sum_{t=0}^{\infty} \gamma^t r_{t+1} \tag{2.1}$$

  where γ represent the discount factor, which determines the magnitude of future accumulative rewards. A reward is local, as it represents the immediate advantage or cost associated with the execution of a specific action, and is independent of the cumulative accomplishments the agent have achieved.
- **Transitions:** Transitions are terms that expresses the characteristics of a system from one state to another state. One transition, also referred to as a cycle or step, comprises four elements (s, a, r, s'), where s represent the current state, a corresponds to the executed action, r reflects the reward and s' denotes the next state.

- **Policy:** A policy, denoted π, is the strategy that the agent deploys to determine which actions to execute in various states, as it maps a particular state to a corresponding action. The optimal policy π^* is the strategy that maximizes the expected reward.
- **Value function:** The value function, denoted $V(s)$, is a function that represents the expected cumulative reward that the agent can receive for a certain state s:

$$V(s) = \mathbb{E}[G_t|S_t = s] \tag{2.2}$$

- **Action-value function:** The action-value function, denoted $Q(s, a)$, is similar to the value function. It is a function that quantifies the expected return that the agent can achieve by performing a certain action a in a particular state s:

$$Q(s, a) = \mathbb{E}[G_t|S_t = s, A_t = a] \tag{2.3}$$

- **Bellman's equation:** Bellman's equation encapsulates the core principles of RL, enlightening why the agent behaves as it does and how it anticipates future rewards:

$$Q(s, a) = \mathbb{E}[r + \gamma Q(s', a')] \tag{2.4}$$

Robot Mouse

To clarify the relationship between the terms introduced above, a relatively simple example can be utilized. For simplicity, the following example examines a robot mouse that navigates in a $4x4$ grid maze.

Example 2.1: Robot Mouse
Consider a robot mouse that is restricted to navigating in a grid maze, where a certain number of cells contain pieces of cheese, and others provide an electric shock, which is illustrated in Figure 2.1.

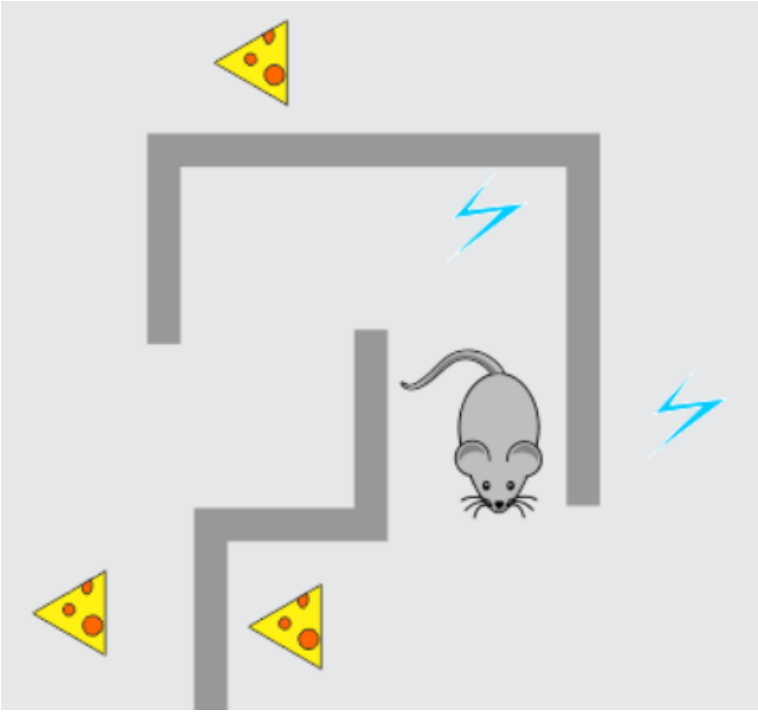

Figure 2.1: *The robot mouse navigating in a maze. [9, p. 3]*

> In this example, the agent is the robot mouse and the environment is the maze containing cheese and electricity. The robot mouse is interacting with the environment by navigating around in the maze, by selecting one of the available actions from the action space $\{up, down, left, right\}$. At each moment, the robot mouse observes the state of the maze to determine which action to perform. Consider the state illustrated on the figure, the robot mouse is limited to the actions up or down. The objective of the robot mouse is to maximize the collected return, where obtaining cheese provides a positive reward, conversely receiving an electric shock incurs a negative reward. Depending on the determined reward function, the robot mouse might even consider accepting an electric shock if it results in a greater overall return.

This example highlights the critical requirement to initially explore the environment, before exploiting the accumulated knowledge. Additionally, it indicates why hard-coding the entire knowledge related to the environment and the optimal sequence of actions is insufficient. This approach is time-consuming and proves impractical when minor adjustments occur in the maze's shape, size, wall configurations, or item placements. As mentioned in Section 1.1, RL exhibits the ability to learn from previous experiences and adapt to complex and dynamic environments.

Q-Learning

To solve the robot mouse problem, Q-learning presents a possible solution because it is a model-free approach, in which prior knowledge of the environment is not required. The agent obtains knowledge by exploring the environment. Q-learning is a foundational algorithm in RL and serves as the basis for more advanced techniques that will be discussed later.

The Q-learning algorithm is a value iteration update of the Bellman equation, following the form:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_a Q(s',a)) \tag{2.5}$$

where α corresponds to the learning rate. The function redefines the action-value function for each time step. To store the information about the environment, the agent generates a searchable database, denoted as a Q-table. Initially, the Q-table is initialized with an arbitrary starting value, such as zero. The Q-table consists of state-action values, with rows representing the various states the agent could encounter, and the columns corresponding to the desired action space.

Utilizing chess notation, the maze's X-axis is labeled with letters from A to D, and the Y-axis with numbers from 1 to 4. The initial location of robot mouse is $C2$, within the maze spanning from $A1$ to $D4$. To clarify, consider the Q-table provided in Table 2.1, where a full table would contain 16 rows and four columns; for simplicity, only the important rows are provided. The robot mouse starts in state $C2$, and decides to navigate up, it encounters an electrical shock, resulting in a penalty of -1. In contrast, if the robot mouse is in state $C1$ and decides to navigate left, it receives a positive reward of $+1$. In the table, an $-$ indicates an infeasible action, while a 0 represents a neutral action.

State/Action	Up	Down	Left	Right
$C1$	0	–	1	0
$C2$	−1	0	–	–

Table 2.1: *Q-table for the robot mouse example.*

As mentioned above, the agent acquires knowledge by interacting with the environment and receiving feedback, in the form of rewards or penalties. The objective is to continually enhance the agent's performance, utilizing the Q-table. As the agent discovers which decisions lead to the most favorable outcomes, the Q-table progressively becomes more reliable, enabling the agent to determine more precise decisions. One of the limitations related to Q-learning is the requirements of a Q-table. In a complex environment with a high-dimensional state space, the Q-table quickly becomes impractically large.

2.2 Deep Q-Network

The DQN approach address several limitations of conventional Q-learning. It was introduced by DeepMind in 2015 [10, p. 1], this approach was a significant innovation in the RL field. It was reaching performance levels that surpassed human capabilities in various Atari 2600 games. The Atari 2600, a home video game console from the late 1970s, contains a variety of old games, that serve as benchmarks for evaluating the performance of RL algorithms. This breakthrough, by employing NN in combination with already existing RL techniques, was the first phase of enhanced capabilities that have contributed to improvements in conventional Q-learning.

The DQN addresses the limitations encountered in the conventional Q-learning approach, when interacting with complex and high-dimensional state-action spaces. One of the major innovations was the incorporation of Q-networks that facilitates the end-to-end learning ability, by approximating the Q-function. The Q-function maps a state to the corresponding Q-values for all feasible actions:

$$Q_\theta : \mathcal{S} \in \mathbb{R}^m \mapsto \mathcal{A} \in \mathbb{R}^n \implies s \mapsto a \tag{2.6}$$

This advantage enables the agent to evaluate the expected future rewards of various actions. Depending on the decision-making technique, the agent would select the action containing the highest Q value. The epsilon-greedy strategy is to balance the trade-off between exploration and exploitation.

Two identically feedforward NN are implemented, an online network ($Q(\theta)$) and a target network ($Q(\bar{\theta})$), where θ and $\bar{\theta}$ are the parameters of the networks respectively. The online network is updated frequently during the learning process, while the target network served as a reference point for updating the parameters. Regularly, the target network will be synchronized with the online network.

Another major innovation was the integration of replay memory (D), which was primarily associated with deep learning, but also served as a crucial element in stabilizing the DQN training process. The replay memory serves the purpose of storing the experienced transitions (s, a, r, s') experienced by the agent during interactions with the environment. The experienced transitions are presumed to have a correlation, so to eliminate this correlation

during training, a uniform batch of random transitions is drawn from the replay memory $((s, a, r, s') \sim U(D))$.

The loss function for the DQN algorithm is computed in terms of the Mean Squared Error (MSE) between the predicted Q-value and the target Q-value:

$$L(\theta) = (r + \gamma \max_{a' \in \mathcal{A}} Q(s', a'; \bar{\theta}) - Q(s, a; \theta))^2 \tag{2.7}$$

The objective is to minimize the loss, and consequently strengthen the DQN agent's decision-making capabilities.

2.3 Double Q-learning

The Double Q-learning (DDQN), introduced in 2016 [10, p. 2095], addresses one of the challenges associated with the DQN approach, the overestimation bias in the Q-values, leading to a decline in the performance. The DDQN approach takes advantage of two distinct networks, a target network and an online network, to decompose the action selection and evaluation process. The loss function for DDQN is defined as:

$$L(\theta) = (r + \gamma Q(s', \arg\max_{a' \in \mathcal{A}} Q(s', a' : \theta); \bar{\theta}) - Q(s, a; \theta))^2 \tag{2.8}$$

In contrast to the DQN approach, where the action with the maximum Q-value was selected, DDQN instead deploys the action corresponding to the maximum Q-value in the current state, and evaluates it by applying the target network. This small change, and still crucial extension, has provided the ability to reducing the overestimation, resulting in an enhanced performance.

2.4 Prioritized Experience Replay

The Prioritized Experience Replay (PER), introduced in 2016 [12, pp. 3-5], was an extension to the already existing replay memory. Instead of uniformly and randomly sampling transitions from the replay memory, the PER approach was to draw the important samples more frequently during the training process. The priority of each sample is calculated on the basis of the TD error (δ). However, a limitation arises from the limited number of samples that update the priorities. A stochastic prioritization is implemented to balance between the greedy prioritization and uniform random sampling, by normalizing the priorities relative to the remaining priorities in the buffer. The probability of sampling transition i is defined as:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \tag{2.9}$$

where $p_i = |\delta_i| + \varepsilon, 0 < \varepsilon \ll 1$ is the priority of transition i, utilizing the proportional prioritization. The parameter α determines the level of prioritization, the smaller the value assigned to α the more importance to prioritize.

The drawback is that this introduces a bias towards recent transitions. To reduce this annealing bias, importance-sampling (IS) weights are utilized.

$$w_i = (\frac{1}{N \times P(i)})^\beta \tag{2.10}$$

where β controls the amount of IS correction over time, reaching $\beta = 1$ at the end of learning. The transitions have been extended to include the variable (w_i), which is the priority weight associated with the transition.

2.5 Dueling Networks

The Dueling networks, introduced in 2016 [15, pp. 4-5], was a reshaping of the existing NN architecture. The Q-network, was decomposed into two elements, the value function $(V(s;\beta) : X \mapsto \mathbb{R})$ and the advantage function $(A(s,a;\alpha) : X \mapsto \mathbb{R}^n)$, where β and α are the parameters of the networks respectively.

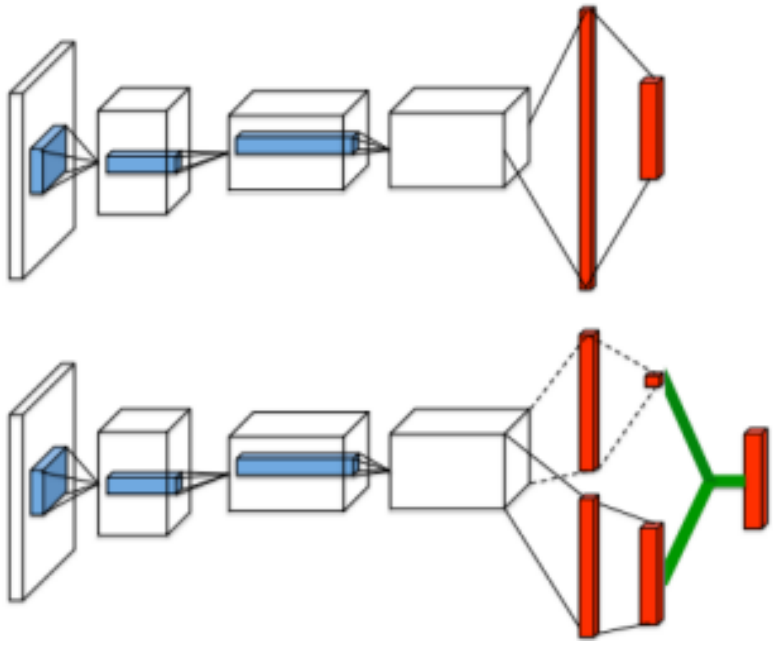

Figure 2.2: *The traditional Q-network (top) and the dueling Q-network (bottom). [15, p. 1]*

The value function represents the reward obtained from state s, while the advantage function indicate how advantageous selecting an action is relative to the others. The combination of the two functions is defined as:

$$Q(s,a;\theta) = V(s;\beta) + A(s,a;\alpha) \tag{2.11}$$

where $\theta := \alpha \cup \beta$. Approximating the two functions separately introduces an identifiable problem. Assuming a predefined Q-value, the individual functions cannot be uniquely determined. To address this issue of identifiability, the highest Q-value is forced to correspond to the value $V(s;\beta)$, implying that the advantage function is equal to zero, and the remaining will be negative:

$$Q(s,a;\theta) = V(s,\beta) + (A(s,a;\alpha) - \max_{a' \in \mathcal{A}} A(s,a;\alpha)) \tag{2.12}$$

The unique recovery is guaranteed, and furthermore this approach can be redefined by replacing the maximum operator with the average:

$$Q(s,a;\theta) = V(s,\beta) + (A(s,a;\alpha) - \frac{1}{|\mathcal{A}|}\sum_{a'} A(s,a;\alpha)) \tag{2.13}$$

In other words, this guarantees that on average the advantage is equal to zero. Consequently, when this is conditional, the agent learns to offset around zero to estimate the advantage for each action.

2.6 Multi-step Learning

The multi-step learning [5, pp. 2-3], in context to DQN, is an extension of the traditional Q-learning approach, where the agent accumulates experience over multiple steps. In a one-step DQN, the agent only considers the immediate reward in addition to the discounted expected reward:

$$Q(s, a; \theta) = r + \gamma \max_{a' \in \mathcal{A}} Q(s', a'; \bar{\theta}) \tag{2.14}$$

The multistep enhancement in DQN enables the agent to incorporate knowledge of future rewards into the learning process, leading to a potential improvement in performance. The multistep DQN for n-step is defined as:

$$Q(s, a; \theta) = R_t^{(n)} + \gamma^{(n)} max_{a' \in \mathcal{A}} Q(S_{t+n}, a'; \bar{\theta}), \quad R_t^{(n)} := \sum_{k=0}^{n-1} \gamma^k r_{t+k+1} \tag{2.15}$$

In multistep learning, an appropriate value for n is important.

2.7 Distributional RL

The distributional RL, introduced in 2017 [1, pp. 8-9], is presumed to be the most complicated expansion of DQN enhancements. It approximates the distribution of the Q-value, instead of representing the mean with a single numerical value. The Distributional RL approach is based on the distributional Bellman equation:

$$Z(s, a) \overset{D}{=} R(s, a) + \gamma Z(s', a'), \tag{2.16}$$

which is almost identical to the traditional Bellman equation, except that $Z(s, a)$ and $R(s, a)$ are now the probability distributions and not a single number.

In the Distributional RL, a predefined support vector covers the entire Q-space, which generally ranges $[V_{min}, V_{max}] \in \mathbb{R}$. The support vector is partitioned into N_{atoms} sub-intervals, with each atom z_i defined as:

$$z_i = V_{min} + i\Delta z \qquad \text{where} \qquad \Delta z = \frac{V_{max} - V_{min}}{N_{atoms} - 1}, \tag{2.17}$$

The probability associated with each atom is defined as:

$$Z_\theta(s, a) = z_i \qquad \text{w.p.} \qquad p_i(s, a) = \frac{e^{\theta_i(s,a)}}{\sum_j e^{\theta_j(s,a)}} \tag{2.18}$$

Employing a discrete distribution presents the challenges, that the Bellman update and the parameterization generally have disjoint supports. To address this challenge, the sample Bellman update $\hat{\tau} Z_\theta$ is projected onto the support Z_θ. Consider a sample transition, (s, a, r, s'), the Bellman update for each atom z_j is expressed as:

$$\hat{\tau} z_j = r + \gamma z_j \tag{2.19}$$

The i^{th} component of the projected update is defined as:

$$(\Phi\hat{\tau}Z_\theta(s,a))_i = \sum_{j=0}^{N-1}\left[1-\frac{|[\hat{\tau}z_j]_{V_{min}}^{v_{max}}-z_i|}{\Delta z}\right]_0^1 p_j(s',\pi(s')) \tag{2.20}$$

where $[\cdot]_a^b$ confines its argument within the range $[a,b]$.

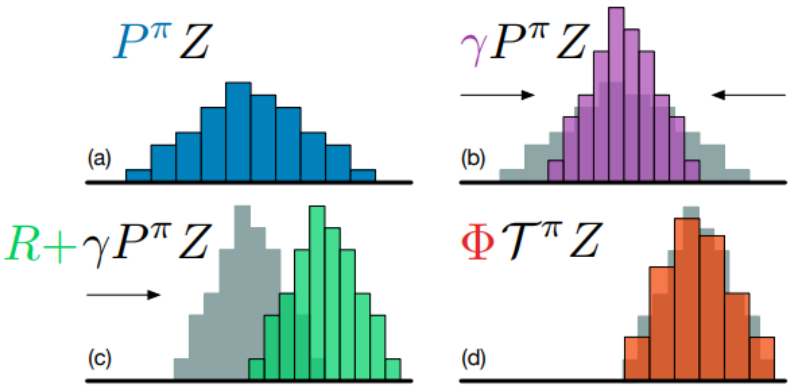

Figure 2.3: *A distributional Bellman operator with a deterministic reward function: (a) Next state distribution under policy π, (b) Discounting shrinks the distribution towards 0, (c) The reward shifts it, and (d) Projection step. [1, p. 2]*

The sample loss for the distributional RL algorithm is computed in terms of the Kullback–Leibler divergence:

$$D_{KL}(\Phi\hat{\tau}Z_\theta(s,a)||Z_\theta(s,a)) \tag{2.21}$$

2.8 Noisy Nets

The noisy nets, introduced in 2018 [4, pp. 8-9], is an enhancement primarily associated with NN, by introducing uncertainty into the NN parameters. For a fully connected layer with a p-dimensional input, the q-dimensional output is typically defined as a linear combination of input and weight:

$$y = wx + b \tag{2.22}$$

where $x \in \mathbb{R}^p$ represents the input of the layer, $b \in \mathbb{R}^p$ corresponds to the bias, and $w \in \mathbb{R}^{q\times p}$ denotes the weight.

Including a small amount of noise to the network parameters, noisy nets encourage exploration in the action space, without requiring additional exploration strategies, such as the epsilon-greedy strategy. For the noisy nets, the traditional output of a perceptron is modified:

$$y = (\mu^w + \sigma^w \odot \varepsilon^w)x + (\mu^b + \sigma^b \odot \varepsilon^b) \tag{2.23}$$

where $\mu^w, \sigma^w \in \mathbb{R}^{q\times p}$ and $\mu^b, \sigma^b \in \mathbb{R}^q$ are learnable parameters and $\odot$ denote the element-wise product. $\varepsilon^w \in \mathbb{R}^{q\times p}$ and $\varepsilon^b \in \mathbb{R}^q$ corresponds to the noise, generated according to two different techniques, independent Gaussian noise or factorized Gaussian noise. The independent Gaussian noise will probably be inefficient, as it requires each weight to have an independent noise. In contrast, the factorized Gaussian noise is more efficient, requiring

only two random variables ε_i and ε_j, to generate noise for each weight:

$$\varepsilon_{i,j}^{w} = f(\varepsilon_i) f(\varepsilon_j) \tag{2.24}$$

$$\varepsilon_{j}^{b} = f(\varepsilon_i) \tag{2.25}$$

where $f(x) = sgn(x)\sqrt{|x|}$. The enhancement of noisy nets enables the agent to discover different and potentially more efficient strategies to interact with the environment. During the training process, the agent learns to ignore noisy terms when exploration is no longer beneficial.

2.9 Rainbow

The Rainbow, introduced in 2018 [6, pp. 1-3], is an advanced variant of the traditional DQN, as it incorporates a variety of enhancements and innovations of techniques that have previously been introduced, to improve the performance and robustness of RL algorithms. In summary, the Rainbow architecture comprises the seven major extensions, each containing their respective beneficial aspects.

- **DQN:** The DQN is the foundational algorithm in the architecture.
- **DDQN:** The DDQN reduces the overestimation of Q-values, by decoupling the action selection and evaluation networks.
- **PER:** The PER improves the sample efficiency by prioritizing significant transitions in the replay memory.
- **Dueling networks:** The dueling networks decomposing the Q-function into a value function and an advantage function for enhanced policy evaluation.
- **Multi-step learning:** The multi-step learning enhances the learning efficiency by bootstrapping over multiple time steps.
- **Distributional RL:** The distributional RL represents the distribution of expected returns, rather than a single scalar value.
- **Noisy nets:** The noisy nets incorporate a noisy stream into the network parameters to improve exploration and robustness.

3 | Federated Learning

In the recent present, the FL approach has been receiving an increased interest in general. The primary objective of this chapter is to outline the essential concepts related to FL, including the fundamental principles, the lifecycle of an FL model, and the typical training process. This chapter is primarily based on [8, pp. 4-9].

3.1 Prerequisites

FL is an ML approach that enables multiple decentralized systems to establish a global model, without exchanging information between them. In contrast to traditional ML techniques, where raw information is transmitted to a centralized system for training, FL facilitates the ability to train models locally. Only model updates are shared with a server, where aggregation is performed to refine the global model. The FL approach emphasizes the training of the collaborative model while maintaining the privacy and security concerns of the data, as mentioned in Section 1.1. In the following, some significant components associated with FL are presented.

- **Clients:** The clients are either devices or nodes that consist of a local dataset, and are participating in the training process by computing model updates according to their respective local datasets. Common examples of clients are cell phones and IoT devices.

- **Aggregation server:** The aggregation sever is the central part of FL that is responsible for gathering the model updates received from the clients. The updates are subsequently aggregated to refine the global model. The Aggregation server is restricted to only receive and processes the model updates gathered from the clients.

- **Global model:** The global model is the shared model that all the clients are attempting to refine. Initially, it is initialized with predefined parameter, and receives periodic updates during the training process. A significant advantage of this approach is the ability to integrate insights from multiple datasets.

3.2 Cross FL

Data privacy preservation has been a longstanding challenge for over five decades. In 2016, a groundbreaking solution, FL, addressed this challenge. Throughout the last decade, FL has actively been implemented into a wide variety of applications that span a range of scales. The FL framework can be divided into two fundamental classes, Cross-silo and Cross-device:

- **Cross-silo:** The Cross-silo FL paradigm is primarily associated to collaborative learning distributed across multiple organizations. The organizations generally have concerns about data privacy and security, preventing them from sharing raw data between the clients and the server. The total amount of client is typically limited, yet they possess substantial computational capabilities. Ensuring reliability is a crucial aspect, as these clients need to be consistently available to refine the global model.

- **Cross-device:** The Cross-device FL framework is primarily centered on collaborative learning distributed among a variety of devices belonging to a single organization. In a cross-device FL system, the number of clients is usually enormous, with the drawback of communication ability being limited. The network of clients is notably prone to unreliability. Through the training process, a considerable fraction of the clients is anticipated to encounter challenges, due to unexpected factors ranging from battery failure to network disruptions.

From the above explanations, it is observed that both the cross-silo and the cross-device FL framework enable collaborative model training across a decentralized environment, while addressing privacy and security concerns. In Table 3.1 a comparison between a traditional centralized ML and the two FL frameworks is presented with various selected characteristics.

	Centralized	Cross-silo	Cross-device
Client's type:	Compute nodes in a single cluster.	Different organizations.	Mobile or IoT devices.
Amount of clients:	1 to 1000.	2 to 100.	There are no restrictions.
Client availability:	All clients.	The majority of all clients.	A fraction of clients.
Data type:	Centralized.	Siloed data, and decentralized.	Decentralized.
Addressability:	Each client is assigned an identity or name.	Each client receives an identity or name.	Client lacks the ability to be indexed.
Reliability:	Relatively few encounters challenges.	Relatively low error rate.	Highly unreliable, anticipated to encounter challenges.
Bottleneck:	Computation.	Computation or communication.	Communication, such as Wi-Fi.

Table 3.1: *Overview of a centralized ML compared with the two FL frameworks. [8, p. 6]*

3.3 Lifecycle Process

The FL lifecycle process, illustrated in Figure 3.1, is commonly initialized when a model engineer identifies a specific challenge, and attempts to develop a novel strategy applicable to this real-world application. To achieve optimality, the model engineer can proceed by developing prototype model architectures and fine-tuning learning hyperparameters within an FL simulated environment that utilizes a virtual dataset as a basis. Next, the FL training process, where an aggregation server administers the process, and the following steps are performed iteratively until a predetermined stopping criterion is satisfied.

- **Client selection:** The client selection is the initial process, where the portion of clients that satisfy the necessary criteria is selected to participate in the training round.

- **Computation:** The computation phase is where the selected clients receive the global model from the server. Subsequently, each client individually computes an adjustment for the global model.
- **Aggregation:** The aggregation server aggregates all the updates that are submitted from the clients, and refines the global model.

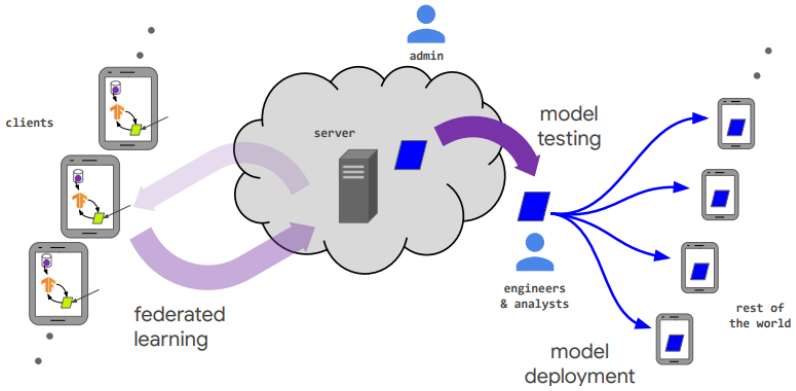

Figure 3.1: *The lifecycle of an FL-trained model and the various actors in a federated learning system. [8, p. 7]*

Several strategies have been developed for the aggregation process [11, p. 10]. One of the simplest and most well-established techniques is the average aggregation (FedAvg). Utilizing this strategy, the aggregation server basically summarizes and averages the received updates. Suppose that there are N clients, each providing an individual update w_i, the aggregated update w is computed as:

$$w = \frac{1}{N} * \sum_{i=1}^{N} w_i \tag{3.1}$$

After a predefined criterion is achieved, the federated model is analyzed and evaluated. This analysis involves the assessment of statistical metrics computed utilizing a standard centralized dataset. In addition, a federated evaluation technique can be employed, where the global model is evaluated on decentralized datasets. In the case that the developed global model is not satisfied, the FL training process is repeated and a new global model is developed.

Once a sequence of analyses and evaluations has been conducted, and an acceptable global model has been established, the standard model launching process begins. The application of the accepted model generally operates independently of the process developed. Additionally, it is essential to consider the compatibility of the model's quality, as the simulated environment will probably be distinct from the real-world application environment.

4 | Experiments

The subject of this chapter is to briefly describe the dateset utilized for training and evaluating the proposed RL framework. Additionally, the chapter will present the framework description there will be considered for this thesis, covering multiple application scenarios. The final section outlines the specification and requirement for the implementation.

4.1 NSL-KDD Dataset

This section is primarily based on [2, pp. 3-4] and [16, pp. 6-8].

In Section 1.1 there was highlighted several crucial reasons why cybersecurity is a paramount aspect in the today's interconnected digital world. For this objective, a relative representative dataset is essential. There exist a limited amount of datasets serving as a benchmark for network Intrusion Detection Systems (IDS) [16, p. 7]. For the thesis, it was determined to utilize the NSL-KDD dataset, serving as a reference point for evaluating the proposed RL framework.

The 2009 NSL-KDD dataset is an enhancement of the well-established KDD Cup99 dataset, which is the primary and most frequently employed dataset for network IDS. The KDD Cup99 dataset comprises approximately 4.900.000 records in the training set and another 3.000.000 records in the testing set, both characterized by 41 features, which will not be discussed further [2, pp. 10-11]. One limitation of the KDD-Cup99 dataset is that there is redundant and duplication of records, actually a proportion of $75 - 78\,\%$ of the records are duplicates, which contributes to an imbalance between the amount of normal traffic and the total number of attacks.

To address this challenge, the NSL-KDD dataset has been refined by eliminating unnecessary and duplicate records in both the training and test datasets, resulting in a remarkably reduced complexity of the dataset. Additionally, this refinement provides several benefits; for instance, the learns performance is not biased towards more frequent records, and the reduced dataset size eliminates the requirement to take a chunk and thereby introduce randomly. Consequently, this ensures more reliable and reproducible results across various research studies.

There are several limitations to consider, especially With a dataset in a sensible area like cybersecurity. As mentioned in Section 1.1, threats are rapidly evolving, and cybercrime is highly profitable, both financially and reputationally. This means that it is impossible to continually maintain the novel attacks and that the datasets can quickly become outdated. For example, this data set was established in 1999, which means that it is already twenty-five years old. Another notable limitation of the dataset is that it does not reflect a universally representative dataset of the real world, as it only includes 40 distinct attack types. However, in reality, the magnitude of potentially threats is never-ending, as novel attacks rapidly are developed.

4.1.1 Analysis

The dataset is freely available from `kaggle.com` [18], and the folder contains the file `index.html`, which provides a description of the NSL-KDD dataset. For the thesis, it was determined to utilize the following two files: `KDDTrain+.txt` and `KDDTest+.txt`. To clarify, the `KDDTrain+.txt` represents the full NSL-KDD training set and `KDDTest+.txt` reflects the complete NSL-KDD testing set; both files include attack-type labels and difficulty level.

The NSL-KDD dataset contains normal and four attack records, named Denial of Service (DoS), Probe, Remote to Local (R2L) and User to Root (U2R).

- **DoS:** The DoS attacks are intended to overwhelm the service with a flood of illegitimate requests, causing significant interruptions. This results in the service being either slow or practically inaccessible to innocent users. A frequently encountered example is the HTTP flood.
- **Probe:** The probe attacks s where network scanning is utilized to capture awareness, with the intention to discover potentially vulnerabilities that can be exploited in subsequent attacks. Some typical functions are port scanning and network sniffing.
- **R2L:** The R2L Attacks is where an attacker obtains unauthorized access to a local system, from a remote machine. This results in access to sensitive information, data breaches, or even compromising data. Common examples are password spoofing and phishing attempts.
- **U2R:** The U2R attacks are where an attacker has a limited access, but subsequently exploits vulnerabilities to obtain root or administrator permissions. Rootkits are a common instance of this kind of behavior.

This dataset comprises a total of 40 distinct attack types, with 23 being employed for training, and 38 for evaluation. The distribution of the datasets employed in the centralized approach is provided in Table A.1 and Table A.2. An analysis of these tables reveals that certain attack types are not included in the training set and, conversely, two attacks types, `spy` and `warezclient`, are omitted from the test set. Additionally, the datasets `KDDTrain+.txt` and `KDDTest+.txt` were mentioned above as sourced from the provided folder. While the datasets, `Train80` and `Test20` were created by the author with the purpose of validating the performance of the trained agent and analyzing the impact of zero-day attacks. For this objective, the original `KDDTrain+` was partitioned into 80 % for training and 20 % for testing, ensuring that both the sets consist of identical attack types.

To train and evaluate the agent's performance, a minimal preprocessing was performed. The NSL-KDD dataset contained three symbolic features; `protocol_type`, `service` and `flag`, which have been removed from the data [16, p. 6]. Additionally, a min-max normalization technique has been employed to normalize all feature values within the range $[0, 1]$.

4.2 Framework Description

In today's interconnected digitized world, the total amount of data traffic transmitted through networks and severs can be significantly, causing it of paramount importance to safeguard the entire network and its components from malicious activity. The purpose of the thesis is to develop a framework for malware detection that prioritizes data privacy preservation. The remaining of this section will present details related to the proposed framework.

4.2.1 Environment

The thesis will examine two distinct environments; centralized and decentralize, each reflecting various real world applications. In the centralized scenario, a potential framework could reflect a single device, such as a laptop, where the malicious detection agent is located. As all data are stored and analyzed on the single device, enhancing privacy and data security by preventing the transmission of sensitive data across a network. Alternatively, if the environment operates as a big data warehouse, which is a centralized database that retrieves and analyzes massive amounts of data from a variety of sources. In this particular scenario, data privacy and security concerns become critical. This centralized infrastructure is vulnerable to attacks, and the retrieve sensitive data is shared with the big data warehouse, increasing the risk of data breaches.

In the decentralized scenario, the framework could be a relatively small and private network consisting of five independently connected devices, where each individual device might potentially encounter distinct types of malicious activity. This exemplifies the FL cross-silo approach, which simulates an environment that reflects the heterogeneous complexities of a modern network architecture. In other words, this framework includes multiply devices, each encompassing an independent dataset to ensure data privacy concerns. The decentralized environment employs an FL framework to refine a global model.

The state space for both scenarios encompasses all feasible stats of the environments, including all available records for each malware class. In the centralized environment, the state space is a large, singular dataset. Consequently, in the decentralized environment, the state space represents the union of the states from all devices. An advantage is the ability to represent the wide variety of malicious activity encounters within a network, with each individual device contributing to the overall state space.

4.2.2 Agent

For the thesis, it was attempted to implement the entire Rainbow RL architecture, nevertheless, because of limitations in time and resources, it was not successfully implemented. Consequently, the proposed RL framework incorporates only the following key elements of the full Rainbow approach: DDQN, dueling networks, and noisy nets. Notable is the implementation of the noisy nets, which features factorized Gaussian noise.

To obtain a reliable comparison to the proposed RL framework, a baseline is established utilizing a conventional DQN, which is presented in Figure 4.1. The proposed RL framework operates similarly to the conventional DQN, with the notable difference by incorporated the previously mentioned enhancements into the DNN.

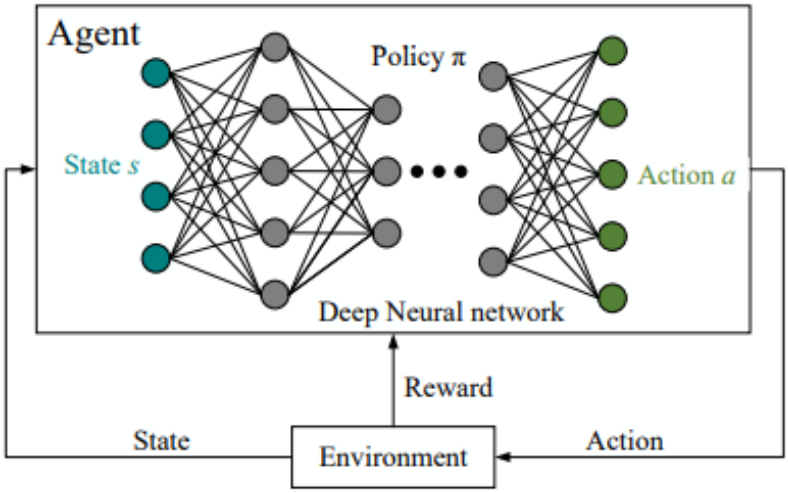

Figure 4.1: *The fundamental components in a DQN. [17, p. 3]*

The action space represents all feasible actions the agent can decide to perform, nevertheless, this thesis considers two distinct strategies; binary and multiclass classification. The binary classification is a simplified classification, where the objective is distinguishing between being attacked or not being attacked. This reflects the crucial question that must be considered in any particular system before an appropriate mitigation strategy is deployed.

The multiclass classification reflects a more realistic representation of the real world, as the awareness that an entity has been subjected to an attack provides a limited actionable amount of knowledge. To attempt to mitigate the attack, the mitigation strategy required sufficient knowledge related to the particular type of attack. Each type of attack aims to achieve distinct objectives, involving various techniques and strategies, and there are countered in distinct various approaches. For the thesis, the action space for the multiclass classification scenario is the five-label presented in Section 4.1.1.

4.3 Implementation and Requirement

The script related to this thesis was written in `Python 3.11`, and has successfully implemented the following components of the Rainbow RL architecture: DDQN, dueling networks and noisy nets. To execute the script, it is required to have minimum `Python 3.10`, as the script utilizes some particular features. The implementation of the Rainbow RL architecture combined with FL was influenced by [3] and [7].

The specifications there have been determined to utilize for the implementation is presented in Table B.1. Additionally, the implementation, training, and evaluation of the proposed RL framework was conducted on a laptop containing an Intel Core i5 CPU.

5 | Results

The purpose of this chapter is to present the results obtained from evaluating the agents' performance in a variety of scenarios associated with multiple applications. For the results, a tabular representation will be provided, where each individual case will be examined three independent times to obtain a more reasonable interpretation of the results. For the seed 2024, an associated illustration in the form of a confusion matrix will be presented. Remark: all the results are presented as a conclusive test, conducted after 1000 episodes were iterated either 100 times for centralized or 50 iterations for the decentralized framework. The results will generally be presented without an additional explanation.

5.1 Baseline

This section establishes a baseline encompassing the scenarios of multiclass classification for the application of a centralized framework. More specifically, the scenarios there will be considered is whether the testing dataset includes zero-day attacks (Zero-day attacks), or not (Common Attacks). The baseline will serve as a reference point for the enhancements discussed later in this chapter.

Common Attacks

Seed	Normal	DoS	Probe	R2L	U2R	Weighted Avg
1806	0.9422	0.999	0.9906	0.9397	1.0	0.9674
1999	0.9471	0.9979	0.9931	0.9648	0.5455	0.9699
2024	0.944	0.9992	0.9897	0.9598	0.7273	0.9684
Average	0.9444	0.9987	0.9911	0.9548	0.7576	0.9686

Table 5.1: *Baseline, where the testing set do not include novel attacks.*

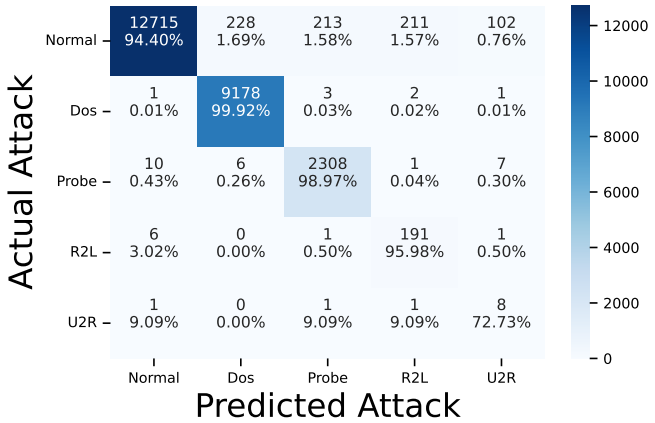

Figure 5.1: *Confusion matrix constructed from the baseline, utilizing seed* 2024.

Zero-day Attacks

Seed	Normal	DoS	Probe	R2L	U2R	Weighted Avg
1806	0.8925	0.829	0.7716	0.0962	0.15	0.7547
1999	0.8854	0.8525	0.7749	0.0741	0.12	0.7567
2024	0.8823	0.8534	0.7464	0.0817	0.135	0.7537
Average	0.8867	0.845	0.7643	0.084	0.135	0.755

Table 5.2: *Baseline, where the testing set includes zero-day attacks.*

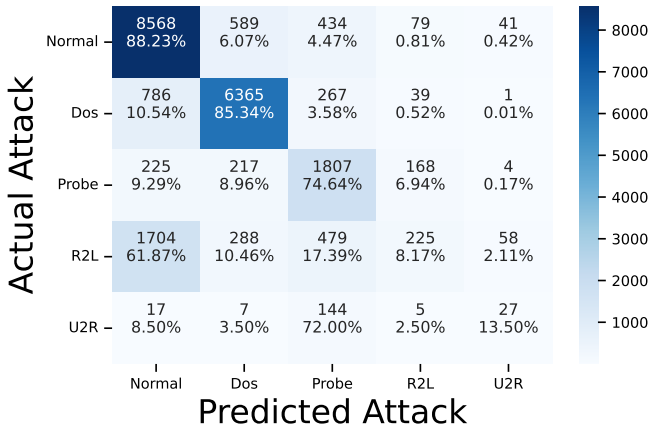

Figure 5.2: *Confusion matrix constructed from baseline for zero-day attacks, utilizing seed 2024.*

Seed	Precision	Recall	F1-score
1806	0.7275	0.7547	0.7254
1999	0.724	0.7567	0.7254
2024	0.7257	0.7537	0.7217
Average	0.7257	0.755	0.7242

Table 5.3: *Performance metrics computed from baseline, including zero-day attacks.*

For the baseline, two tables and two confusion matrices are provided. The results for the common attacks are provided in Table 5.1, where horizontally the table present the detection rate, also referred to as recall, obtained from the three independent seeds 1806, 1999 and 2024. In general, the performance of the agent is stable, as the variation across the three seeds is minimum. The only notable result is the U2R, as it results in three different values.

The results for the zero-day scenario are presented in Table 5.2, where similar patterns are observed across all three seeds. Normal and Dos contribute to the highest detection rates, followed by Probe. Unfortunately, there is a significant reduction in detection rates for R2L and U2R attacks, both being reduced below 15 %. In Table 5.3, the three metrics; precision, recall and F1-score is presented, where the lowest is approximately 72 %.

5.2 Rainbow Components on Centralized Framework

This section presents the results of incorporating the Rainbow components into a centralized environment. The results follow similar to the baseline.

Common Attack

Seed	Normal	DoS	Probe	R2L	U2R	Weighted Avg
1806	0.9559	0.9989	0.9931	0.9548	0.7273	0.9749
1999	0.9533	0.9985	0.9919	0.9548	0.5455	0.9732
2024	0.9537	0.9989	0.9927	0.9397	0.5455	0.9735
Average	0.9543	0.9988	0.9926	0.9498	0.6061	0.9739

Table 5.4: *Results obtained from the Rainbow components, without including novel attacks.*

Zero-day Attack

Seed	Normal	DoS	Probe	R2L	U2R	Weighted Avg
1806	0.909	0.8057	0.684	0.1554	0.065	0.7511
1999	0.8983	0.7974	0.589	0.1267	0.135	0.7307
2024	0.8981	0.8469	0.8534	0.0868	0.085	0.77
Average	0.9018	0.8167	0.7088	0.123	0.095	0.7506

Table 5.5: *Results obtained from the Rainbow components, including zero-day attacks.*

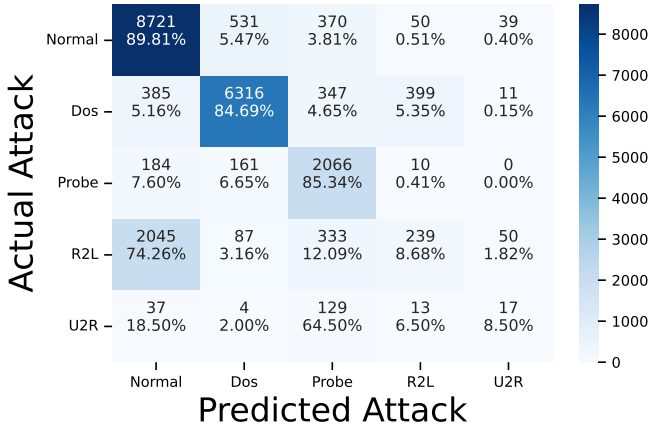

Figure 5.3: *Confusion matrix constructed from the Rainbow components, including zero-day attacks and utilizing seed* 2024.

Seed	Precision	Recall	F1-score
1806	0.7295	0.7511	0.7277
1999	0.7322	0.7307	0.7027
2024	0.7354	0.77	0.7396
Average	0.7324	0.7506	0.7233

Table 5.6: *Performance metrics computed from the Rainbow components, including zero-day attacks.*

The results, obtained for the centralized framework utilizing the Rainbow components, look with the first glance similar to the baseline. For the common attacks, there is a not significant improvement, which could have be caused by randomness.

5.3 Zero-day Attacks in a Federated Framework

This section outlines the results of deploying the Rainbow components into a FL framework, while still encountering zero-day attacks. Two distinct data distribution are considered: IID and non-IID. In the IID scenario, the data is divided equally between the clients in the FL framework. Consequently, in the non-IID scenario, one of the five devices encounters all state randomly, while the remaining four devices are constantly subject to malicious behavior.

IID

Seed	Normal	DoS	Probe	R2L	U2R	Weighted Avg
1806	0.8965	0.9102	0.7158	0.0897	0.095	0.7759
1999	0.8962	0.8155	0.8765	0.069	0.185	0.76
2024	0.8994	0.8116	0.6386	0.0828	0.135	0.7358
Average	0.8974	0.8458	0.7436	0.0805	0.1383	0.7572

Table 5.7: *Results obtained from an IID FL framework, utilizing the Rainbow components.*

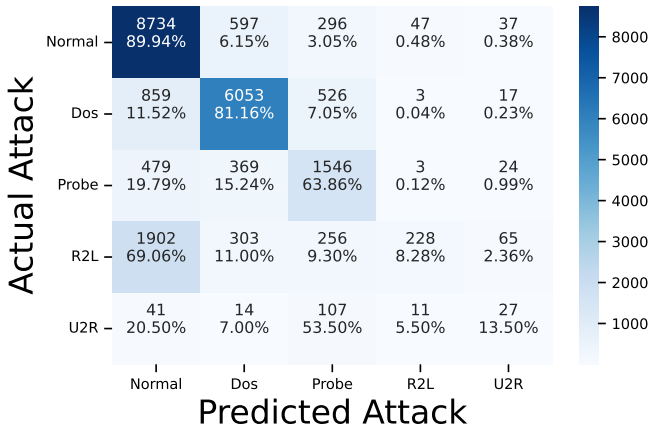

Figure 5.4: *Confusion matrix constructed from the IID FL framework, including zero-day attacks and utilizing seed* 2024.

Seed	Precision	Recall	F1-score
1806	0.7631	0.7759	0.7406
1999	0.779	0.76	0.7273
2024	0.7437	0.7358	0.7011
Average	0.7619	0.7572	0.723

Table 5.8: *Performance metrics obtained from the IID FL framework.*

Non-IID

Seed	Normal	DoS	Probe	R2L	U2R	Weighted Avg
1806	0.8743	0.8773	0.7299	0.0926	0.19	0.7582
1999	0.8756	0.847	0.7402	0.1195	0.115	0.7525
2024	0.8636	0.8923	0.6948	0.1064	0.155	0.7562
Average	0.8712	0.8722	0.7216	0.1062	0.153	0.7556

Table 5.9: *Results obtained from non-IID FL framework, utilizing the Rainbow components.*

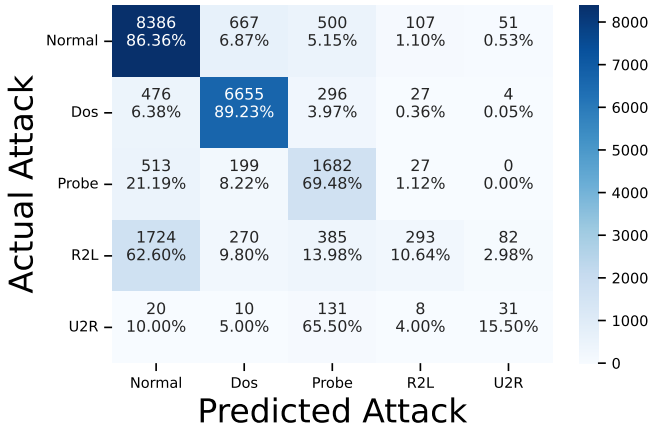

Figure 5.5: *Confusion matrix constructed from the non-IID FL framework, including zero-day attacks and utilizing seed* 2024.

Seed	Precision	Recall	F1-score
1806	0.7435	0.7582	0.7325
1999	0.7299	0.7525	0.7311
2024	0.7465	0.7562	0.7259
Average	0.74	0.7556	0.7298

Table 5.10: *Performance metrics obtained from the non-IID FL framework.*

The results for the IID scenario are provided in Table 5.7, presenting the best individual evaluation and the highest average result for the thesis. The R2L and U2R attacks are preventing from further enhancement, because of the low detection rate. In Figure 5.4 the distribution of the predicted attack is displayed, where it is indicated that 69 % of the R2L attack is predicted as normal.

In the non-IID scenario, where only one of the clients encounters the normal state, the global model successfully manages to captures it, with only a slight decrease in detection rate. The patterns remain consistent, as Normal and DoS achieve the highest detection

rates, followed by Probe, while R2L and U2R attacks have the lowest detection rates, with a maximum individual detection rate of 20 %.

5.4 Comparative study

To quantifying the results of the proposed RL frameworks, it will be compared with well-established classification models. In [16, p. 13], the authors have developed a table for comparison of detection rates of a variety of classification models, there have frequently been employed in the literature for IDS. In Table 5.11 a subset of the classification models are provided for the `KDDTest+` dataset, inclusive the author's proposed model, SAVAER-DNN.

Classification	Normal	DoS	Probe	R2L	U2R	Weighted Avg
K-Nearest Neighbor	0.9275	0.8225	0.594	0.1467	0.03	0.666
Support Vector Machine	0.9282	0.7485	0.6171	0.0	0.0	0.5673
Random Forest	0.9332	0.7580	0.5894	0.1082	0.01	0.6051
DNN	0.9529	0.8315	0.6278	0.0766	0.05	0.7082
SAVAER-DNN	**0.953**	**0.851**	**0.7447**	**0.5359**	**0.445**	**84.86**
This thesis	0.8974	0.8458	0.7436	0.0805	0.1383	0.7572

Table 5.11: *Comparative analysis of dectection rates of a variety of classification methods on the NSL-KDD dataset. [16, p. 13]*

From the Table 5.11, it is observed that the author's proposed model is superior, exhibiting the highest detection rates for each malware class. Additionally, this thesis falls only slightly behind in two classes: DoS and Probe. Lastly, as mentioned in this thesis, the detection rates for R2L and U2R remain challenging. The table indicates that other well-established classification models also fall short in detecting these classes.

6 | Conclusion

In the rapidly evolving landscape of cybersecurity, malware detection provides a complex and dynamic challenge. This thesis explores innovative techniques in the reinforcement learning and the federated learning domain to enhance the robustness and sophistication of malware detection systems. The reinforcement learning framework, with its ability to learn from previous experiences and adapt to evolving attack surfaces and the federated learning framework, facilitates collaborative model training across decentralized environments while maintaining data privacy and confidentiality, offer potentially enhancements in the cybersecurity domain.

This thesis has developed a reinforcement learning framework, which incorporates components of the Rainbow reinforcement learning approach, there has demonstrated superior results in Atari games and similar potential advancements could be achieved in the realm of malware detection. To quantifying the results of the proposed reinforcement learning framework, a baseline utilizing the conventional deep Q-network was established. The purposed framework was evaluated in a centralized and decentralized environment. For the thesis it was determined to utilize the well-established dataset NSL-KDD, which comprises a total of 40 distinct attack types, with 23 being employed for training, and 38 for evaluation.

The results obtained remain consistent, as Normal and DoS achieve the highest detection rates of $85 - 90\,\%$, followed by Probe $75\,\%$, while R2L and U2R attacks have the lowest detection rates, with a maximum individual detection rate of $20\,\%$. From the results it was clear that zero-day attacks remain a significant challenge to the cybersecurity domain, as this type of attacks includes novel and unseen malware, where conventional cybersecurity approaches fall short. The results obtained without novel attacks included, provided detection rates for Normal, DoS, Probe and R2L with above $95\,\%$, while U2R has a detection rate of $60\,\%$.

Bibliography

[1] Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. pages 449–458, 2017.

[2] Sarika Choudhary and Nishtha Kesswani. Analysis of kdd-cup'99, nsl-kdd and unsw-nb15 datasets using deep learning in iot. Procedia Computer Science, 167, 2020.

[3] Curt-Park. rainbow-is-all-you-need. URL `https://github.com/Curt-Park/rainbow-is-all-you-need`. Accessed: 03-06-2024.

[4] Shuai Han, Wenbo Zhou, Jing Liu, and Shuai Lü. Nrowan-dqn: A stable noisy network with noise reduction and online weight adjustment for exploration. arXiv preprint arXiv:2006.10980, 2020.

[5] J Fernando Hernandez-Garcia and Richard S Sutton. Understanding multi-step deep reinforcement learning: A systematic study of the dqn target. arXiv preprint arXiv:1901.07510, 2019.

[6] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. 32(1), 2018.

[7] jan kreischer. Fedrl-for-iot-security. URL `https://github.com/jan-kreischer/FedRL-for-IoT-Security`. Accessed: 03-06-2024.

[8] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. Foundations and trends® in machine learning, 14(1–2), 2021.

[9] Maxim Lapan. Deep Reinforcement Learning Hands-On. Packt, 2020.

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. nature, 518(7540), 2015.

[11] Mohammad Moshawrab, Mehdi Adda, Abdenour Bouzouane, Hussein Ibrahim, and Ali Raad. Reviewing federated learning aggregation algorithms; strategies, contributions, limitations and future perspectives. Electronics, 12(10), 2023.

[12] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. arXiv preprint arXiv:1511.05952, 2015.

[13] Ashish Kumar Shakya, Gopinatha Pillai, and Sohom Chakrabarty. Reinforcement learning algorithms: A brief survey. Expert Systems with Applications, 2023.

[14] Ajeet Kumar Sharma, Rakesh Kr Galav, and Bhisham Sharma. A comprehensive survey of various cyber attacks. 2023.

[15] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. 2016.

[16] Yanqing Yang, Kangfeng Zheng, Bin Wu, Yixian Yang, and Xiujuan Wang. Network intrusion detection based on supervised adversarial variational auto-encoder with regularization. IEEE access, 8:42169–42184, 2020.

[17] Junkai Yi and Xiaoyan Liu. Deep reinforcement learning for intelligent penetration testing path design. Applied Sciences, 13(16), 2023.

[18] M. HASSAN ZAIB. Nsl-kdd. URL `https://www.kaggle.com/datasets/hassan06/nslkdd`. Accessed: 03-06-2024.

Appendices

A | Distribution of the dataset

Class	Attack	Validation		Evaluating	
		Train80	Test20	KDDTrain+	KDDTest+
Normal	normal	53.874	13.469	67.343	9711
DoS	back	765	191	956	359
	land	14	4	18	7
	neptune	32.971	8.243	41.214	4657
	pod	161	40	201	41
	smurf	2.117	529	2.646	665
	teardrop	714	178	892	12
	apache2	–	–	–	737
	mailbomb	–	–	–	293
	processtable	–	–	–	685
	udpstorm	–	–	–	2
Subtotal		36.742	9.185	45.927	7.458
Probe	ipsweep	2.879	720	3.599	141
	nmap	1.194	299	1.493	73
	portsweep	2.345	586	2.931	157
	satan	2.906	727	3.633	735
	mscan	–	–	–	996
	saint	–	–	–	319
Subtotal		9.324	2.332	11.656	2.421

Table A.1: *The class distribution of the NSL-KDD dataset. [16, p. 8]*

Class	Attack	Validation		Evaluating	
		Train80	Test20	KDDTrain+	KDDTest+
R2L	ftp_write	6	2	8	3
	guess_passwd	42	11	53	1.231
	imap	9	2	11	1
	multihop	6	1	7	18
	phf	3	1	4	2
	spy	2	0	2	–
	warezclient	712	178	890	–
	warezmaster	16	4	20	944
	named	–	–	–	17
	sendmail	–	–	–	14
	snmpgetattack	–	–	–	178
	snmpguess	–	–	–	331
	worm	–	–	–	2
	xlock	–	–	–	9
	xsnoop	–	–	–	4
Subtotal		796	199	995	2.754
U2R	buffer_overflow	24	6	30	20
	loadmodule	7	2	9	2
	perl	2	1	3	2
	rootkit	8	2	10	13
	httptunnel	–	–	–	133
	ps	–	–	–	15
	sqlattack	–	–	–	2
	xterm	–	–	–	13
Subtotal		41	11	52	200
Total		100.777	25.196	125.973	22.544

Table A.2: *Continuation of Table A.1.*

B | Specification

Specifications	
Seeds	$1806, 1999, 2024$
DQN architecture	$(38, 64, 32, 5)$
Activation function	$ReLU$
Loss function	MSE
Optimizer	$Adam$
Learning rate α	0.001
Batch size	32
Explore factor ε	1
ε_decay	0.995
ε_ min	0.01
Discount factor γ	0.01
Reward function	$(1, -1)$
# Episodes	1000
# Training sessions	100 or 50

Table B.1: *Specifications for the implementation.*