Journal of the Brazilian Computer Society, 2026, 32:1, doi: 10.5753/jbcs.2026.5365

Enhancing Red Team Agent Learning with the Kill Chain Catalyst Algorithm in Capture the Flag Scenarios

Antonio Horta [**Military Institute of Engineering, Accenture** | ***antonio@horta.net.br***]
Anderson dos Santos [**Military Institute of Engineering, Venturus Centro de Inovação Tecnológica** | ***anderson@ime.eb.br***]
Ronaldo Goldschmidt [**Military Institute of Engineering** | ***ronaldo.rgold@ime.eb.br***]

Military Institute of Engineering, IME. Praça General Tibúrcio, 80, Urca, Rio de Janeiro, RJ, 22290-270, Brazil.

Received: 21 December 2024 • **Accepted:** 20 June 2025 • **Published:** 16 March 2026

Abstract. With the advancement of technology, tasks once performed by humans have increasingly transitioned to machines or agents equipped with artificial intelligence, including various cyber security domains. From the perspective of real-world cyber attacks, executing actions with minimal failures and steps is critical to reducing the likelihood of exposure. Although research on autonomous cyber attacks predominantly employs Reinforcement Learning (RL), this approach has gaps in scenarios such as limited training data, low resilience in dynamic environments, and limited interpretability of decision-making policies. Therefore, Kill Chain Catalyst (KCC), an *RL* algorithm based on Gini Impurity-Based Weighted Random Forest that prioritizes interpretability, efficiency in scenarios with limited experience, and resilience in dynamic environments explored by *RL* agents, has been introduced. *KCC* leverages decision tree logic for enhanced interpretability and employs a catalyst module inspired by genetic alignment to optimize the search for efficient attack sequences. More than 150 attack experiments were conducted to evaluate learning in terms of offset, speed, and generalization. The analysis focused on the steps, rewards, and failures of agents using the RL algorithms *KCC*, *PPO*, *DQN*, *TRPO*, and *A2C*, within a *Capture the Flag* tournament setting. Both static and dynamic scenarios with limited learning experiences were considered. These experiments demonstrate the superior performance of *KCC*, revealing differences of up to 198.69% for steps, 129.43% for rewards, and 1096.39% for failures when performing attacks using *KCC* compared with the other algorithms.

Keywords: Autonomous Red Team Agents, Reinforcement Learning, Kill Chain, Genetic Align, Random Forest

1 Introduction

With the advancement of technology, tasks once performed by humans are now autonomously executed by machines embedded with artificial intelligence [Thangavel *et al*., 2024; Sharma and Rana, 2024]. Beyond industrial robots and autonomous vehicles, studies in the field of cyber attacks aim to automate red team operations [Al-Azzawi *et al*., 2024; Paudel and Amariucai, 2023], enhancing security for companies, training exercises, Capture the Flag (CTF) competitions, and cyber warfare.

In cyber attacks, particularly in scenarios such as *CTF* tournaments, attackers operate without prior knowledge of the target environment, which unfolds dynamically during the campaign [Ortiz-Garces *et al*., 2023]. According to Che Mat *et al*. [2024], stealth becomes a relevant aspect, underscoring the importance of a strategic and efficient sequencing of attack actions to minimize steps and failures, thereby reducing the likelihood of exposure. Given the extensive damage caused by cyber attacks in recent years, the study of autonomous attack strategies has become increasingly significant.

Reinforcement Learning (RL) has emerged as a widely adopted approach in attack automation [Gangupantulu *et al*., 2021; Pozdniakov *et al*., 2020; Chen *et al*., 2023; Zhou *et al*., 2021; Yang and Liu, 2022; Tran *et al*., 2021; Standen *et al*., 2021; Li *et al*., 2022]. *RL* employs a trial-and-error process in which an agent learns from its environment through rewards and penalties, progressively improving its performance over time. However, the effectiveness of an attack hinges on achieving objectives such as resilience in dynamic environments, for example, where a victim might remediate vulnerabilities during the attack, and stealth, which involves minimizing steps and failures during the incursion. Existing studies [Chen *et al*., 2023; Li *et al*., 2022; Holm, 2022; Zhou *et al*., 2021; Yang and Liu, 2022; Tran *et al*., 2021; Standen *et al*., 2021] predominantly prioritize maximizing rewards when evaluating algorithms, often neglecting aspects such as stealth and resilience. Despite the potential of *RL* agents in autonomous cyber attacks, Horta *et al*. [2024a] highlights the limitations, including high failure rates, poor efficiency with limited training data, and low resilience in dynamic scenarios. Furthermore, *RL* agents often rely on neural networks as the core learning engine, which requires large amounts of data from their experiences to understand the environment and achieve objectives. This dependency on vast datasets, combined with the complexity of explaining the decisions made by these agents, presents additional issues to investigate. Horta *et al*. [2024b] introduced the Kill Chain Catalyst (KCC) to address limitations in the use of *RL* for cyber attacks in the state of the art. The *KCC* incorporates a catalyst inspired by genetic alignment to optimize the search for effective attack chain sequences in which combined with a decision tree-based

logic, outperformed all other algorithms in *RL*-driven attack scenarios. *KCC* leverages decision tree logic to guide agents in attacks, enhancing resilience in dynamic environments, stealth by minimizing failures and steps, and effectiveness in scenarios with limited data, while also enabling explainability of decisions. The standout feature of *KCC* in sequential decision-making problems for cyber attacks lies in its use of *Gini Impurity-Based Weighted Random Forest* (GIWRF) as the *RL* engine, in contrast to other algorithms that are neural network-based.

This work builds upon the findings of the original article [Horta *et al*., 2024b], aiming to refine the analysis previously conducted to address the gaps related to how evaluating the learning and improving the experiments. The proposed extension investigates the learning process of agents within the same context as the original study, using the *KCC* algorithm to not only minimize failures and steps with limited data but also to be resilient in dynamic environments. The focus is on a more detailed analysis of attack sequencing learning, with an emphasis on its applicability in *CTF* scenarios, based on the execution of a larger set of experiments.

Experiments were conducted to showcase the use of *KCC*, along with a comparison and explanation of their applicability in different situations. The experiments were performed to analyse the learning curve in terms of steps, rewards, and failures for agents using the RL algorithms *KCC*, *PPO*, *DQN*, *TRPO*, and *A2C* in the context of a *CTF* tournament for static and dynamic scenarios with limited learning experiences. These experiments demonstrate the superior performance of *KCC*, revealing differences of up to 198.69% for steps, 129.43% for rewards, and 1096.39% for failures when performing attacks using *KCC* compared with algorithms such as *A2C*, *PPO*, *TRPO*, and *DQN*, which struggled to generalize attack sequences.

As contributions, beyond the primary *KCC* agent itself, this article consolidates the execution of over 150 attack emulation experiments, providing access to a repository containing all experiments, obtained results, the testing framework, and source code to replicate the experiments or conduct new ones. In addition, this work delves into the concepts of *offset*, *speed*, and *generalization* as specific metrics for evaluating agent learning curves in attack contexts. A dedicated method for calculating each of these metrics is also presented, enabling the objective comparison of different agents' learning processes in attack scenarios.

The article is organized to provide a comprehensive examination of the topic. The introduction outlines the research objectives, followed by a background section detailing key concepts used in this study, including Sequence Alignment inspired by genetic processes, an overview of *RL*, Random Forest, *GIWRF*, and Learning Curves Analysis. The related work section contextualizes the research through a review of relevant literature. The methodology section elaborates on the *KCC* approach and the method for comparing the learning curves of attacks, while the experiments and results section provides an in-depth analysis of the experimental setup and findings. Finally, the conclusion synthesizes the outcomes and suggests directions for future research.

2 Background

The background section delves into foundational concepts to the study, including *Sequence Alignment* from genetic alignment, *Reinforcement Learning*, *Random Forest*, *Gini Impurity-Based Weighted Random Forest*, and *Learning Curves Analysis*.

2.1 Sequence Alignment

In the study of variations between genes, proteins, or organisms, sequence alignments serve as a valuable tool for predicting structural relationships, functions, and evolutionary changes. These alignments involve the comparison of two or more DNA or protein sequences, evaluating their similarity [Ibrahim *et al*., 2024].

The FASTA format, as outlined by Poinsignon *et al*. [2023], provides an efficient and concise representation of biological sequences such as DNA, RNA, and proteins. This format presents sequences as character strings, where each character corresponds to a nucleotide or an amino acid. Due to its simplicity and compatibility, FASTA has gained widespread adoption in bioinformatics, particularly for tasks involving sequence alignment and comparison.

Proteins, which consist of amino acids represented by single-letter abbreviations, have a significant role in genetic alignment. The sequence of amino acids within a protein dictates its function and structure. To compare and align protein sequences, the *Needleman-Wunsch* algorithm, highlighted by Gancheva and Stoev [2023], is commonly employed. According Ibrahim *et al*. [2024], this dynamic programming technique seeks the optimal global alignment between two sequences using a scoring system. The algorithm constructs a matrix to store alignment scores and identifies the best alignment through the matrix. The scoring system incorporates match, mismatch, and gap penalties to effectively assess sequence similarity.

The Needleman-Wunsch algorithm is defined by Equation 1, where S is the score matrix, Equation 2, where a_i and b_j are the elements of the two sequences being aligned, and d is the gap penalty:

$$S(i,j) = \max \begin{cases} S(i-1,j-1) + \text{score}(a_i, b_j) \\ S(i-1,j) - d \\ S(i,j-1) - d \end{cases} \tag{1}$$

$$\text{score}(a_i, b_j) = \begin{cases} \text{match} & \text{if } a_i = b_j \\ \text{mismatch} & \text{otherwise} \end{cases} \tag{2}$$

As outlined by Aspland *et al*. [2021], Needleman-Wunsch algorithm perform alignment between proteins $P1$ and $P2$, allowing users to specify match, mismatch, and gap values. The most common choices for these values are Match $(m = 1)$, Mismatch $(s = -1)$, and Gap $(g = -1)$.

By employing the *Needleman-Wunsch* algorithm, researchers can identify conserved regions and structural similarities among any kind of sequences, specially, proteins, aiding in the understanding of their biological functions and evolutionary relationships.

Table 1 illustrates an example of sequence alignment through the *Needleman-Wunsch* algorithm. Each row corresponds to a distinct DNA sequence, identified by an *Id*, displaying the original sequence and its aligned counterpart generated during the alignment process. The aligned sequences are presented in a monospaced font to facilitate the visualization of gaps and matching bases. The last row showcases the *consensus* sequence, derived from aligning the given sequences. This consensus sequence represents the most prevalent base at each position, offering insights into shared characteristics among the aligned sequences.

Table 1. Example for sequence align by Needleman-Wunsch

Id	Sequence	Align
Seq1	AGTACGTA	A-GTACGT-A
Seq2	ACTACGTA	AC-TACGT-A
Seq3	ACGTATT	ACGTA--TT-
Seq4	ACGTACGTT	ACGTACGTT-
Seq5	ACGTACGTC	ACGTACGT-C
Consensus	-	ACGTACGTTA

2.2 Reinforcement Learning

RL is a subfield of machine learning where an agent learns to make decisions by interacting with an environment to maximize cumulative rewards. This learning paradigm is characterized by the agent receiving feedback from the environment in the form of rewards or punishments, which it uses to update its knowledge and improve future actions. Sutton and Barto [2018] provided a comprehensive framework for understanding RL, introducing key concepts such as the Markov Decision Process (MDP), value functions, and policy optimization. In their seminal work, they describe how an *RL* agent iteratively refines its policy by balancing exploration of new actions and exploitation of known rewarding actions, ultimately aiming to find the optimal policy that maximizes the expected cumulative reward over time.

Furthermore, *RL* can be employed in fully or partially observable environments. In the case of situations where the agent has complete visibility of the environment, *MDP* is used, and for situations where the agent has no knowledge of the environment and how far it is to go to achieve its goals, *Partially Observable MDP* (POMDP) is employed.

RL algorithms include: Deep Q-Network (DQN) [Mnih *et al*., 2015], Proximal Policy Optimization (PPO) [Schulman *et al*., 2017], Trust Region Policy Optimization (TRPO) [Schulman *et al*., 2015] and Actor-Critic (A2C) [Mnih *et al*., 2016].

2.3 Random Forest

Random Forest, a prominent supervised machine learning algorithm, is recognized for its versatility in addressing regression and classification tasks [Breiman, 2001]. This algorithm operates through an ensemble of decision trees, each constructed independently and drawn randomly from the training dataset. The intrinsic strength of *Random Forest* lies in its ability to mitigate overfitting tendencies present in individual trees, thereby enhancing generalization performance.

Decision trees, integral to the *Random Forest* framework, are recursive binary structures designed for optimal feature discrimination. The randomness introduced during the construction of decision trees in *Random Forest* is an important feature. During training, each tree is grown using a bootstrap sample, ensuring diversity among the trees. Moreover, at each split in the tree, a random subset of features is considered, preventing the dominance of any single feature.

In the context of classification tasks, *Random Forest* employs measures such as *entropy* or *Gini impurity* to determine the optimal feature for node splits. *Entropy* measures the information gain, while *Gini impurity* assesses the probability of misclassification. The algorithm chooses the split that maximally reduces *entropy* or *Gini impurity*, promoting effective feature discrimination [Farouk *et al*., 2024].

2.3.1 Gini Impurity-Based Weighted Random Forest

According Disha and Waheed [2022], *Random Forest* is an ensemble classifier constructed from multiple decision trees, incorporating various feature importance metrics. In this sense, *Gini Impurity-Based Weighted Random Forest* is an approach for feature selection. One such metric involves deriving the importance score through the training of the classifier. Moreover, traditional machine learning classification algorithms assume that all classes in the training set possess similar importance, constructing models without accounting for potential imbalances in class distribution within the training data. In order to assess the relevance of features in the context of imbalanced data, this feature selection technique introduces a weight adjustment mechanism within the *Random Forest* algorithm, contingent on the *Gini impurity*, $i(\tau)$. The *Gini impurity* gauges the efficacy of a split in dividing total samples of binary classes at a specific node τ, such as Equation 3.

$$i(\tau) = 1 - p_p^2 - p_n^2, \tag{3}$$

where p_p represents the fraction of positive samples and p_n denotes the fraction of negative samples among the total number of samples (N) at node τ. Moreover, the Gini impurity reduction derives from any optimal split $\Delta i_f(\tau, M)$ in which is gathered together for all the nodes τ in the M number of weighted trees in the forest, individually for all of the features (Equation 4).

$$I_g(f) = \sum_M w_{p,n} \sum_\tau \Delta i_f(\tau, M) \tag{4}$$

where I_g is the *Gini* importance, which specifies the frequency of a particular feature (f) being selected for the split. The significance of the feature's overall discriminative value for the binary classification task. Assigning the weight $w_{p,\,n}$ defines the imbalanced class distribution in the learning algorithm. The weight adjustment for positive class $w_p = \frac{n_n}{N}$, and for negative class $w_n = \frac{n_p}{N}$. Considering, $w_p + w_n = 1$ and for imbalanced class data $w_p \neq w_n$. The number of negative instances is represented as n_n and the positive instances are denoted as n_p, and N is the total number of instances in the training dataset.

2.4 Learning Curves Analysis

Learning, in the context of machine learning, represents a process whereby agents attempt to interpret patterns within data, given limited or no prior knowledge of the environment. Agents, without intrinsic knowledge of their operational domain, engage in a training phase aimed at optimizing their internal parameters to predict responses accurately in unseen scenarios. According to Viering and Loog [2023], the learning curve is a graphical representation that delineates the agent's generalization performance, measured as its accuracy or error, as a function of the number of training instances. These curves facilitate the understanding of how different factors influence an agent's performance and guide the evaluation of learning dynamics across different model types.

According to Da Silva and Costa [2019], the metrics *jumpstart* (offset), *time to threshold* (speed), and *asymptotic performance* (generalization) are central in the analysis of learning curves. The offset refers to the initial performance level when training begins, which may vary substantially depending on the inherent complexity of the model and its baseline capabilities. Speed indicates the rate at which the model improves its accuracy or reduces its error during training. A higher learning speed reflects more efficient learning, which is typically achievable with simpler models or well-structured data. Generalization, conversely, measures the model's capability to maintain its predictive accuracy when exposed to new data, signifying the true extent of the agent's learning.

Figure 1 shows the learning curves for these three metrics over ten periods, where *Agent A* starts with an offset advantage in initial knowledge compared to *Agent B*. Throughout the training, *Agent A* also reaches the threshold in less time. Finally, both agents achieve asymptotic performance at the end of training.

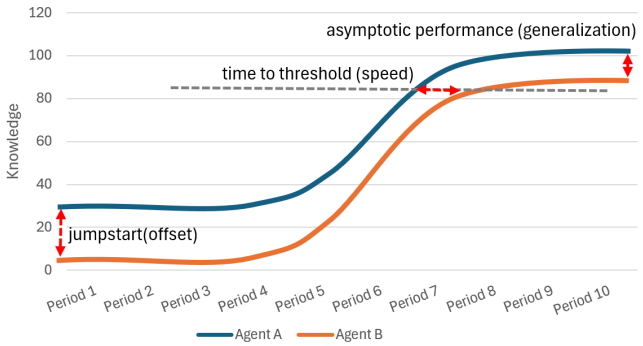


Figure 1. Learning curves.

These metrics serve as indicators for comparing learning performances across diverse models and datasets. They allow for empirical and theoretical analysis of learning curve shapes, further supporting model selection, hyperparameter tuning, and resource allocation during model training [Viering and Loog, 2023].

3 Related Work

The literature review conducted for this research underscores the prevalent use of *RL* in automating cyber attacks [Chen *et al*., 2023; Li *et al*., 2022; Zhou *et al*., 2021; Yang and Liu, 2022; Tran *et al*., 2021; Standen *et al*., 2021]. However, the identified studies predominantly rely on simulators [Zhou *et al*., 2021; Yang and Liu, 2022; Tran *et al*., 2021; Standen *et al*., 2021] or directly exploit vulnerabilities listed in the Common Vulnerabilities and Exposures (CVE) database on real hosts [Chen *et al*., 2023]. In such scenarios, these attacks are typically categorized as post-breach, as they bypass reconnaissance tactics aimed at gathering environmental information like usernames, access credentials, or passwords.

Consequently, in the literature review, except for Li *et al*. [2022], the kill chains produced by *RL* agents primarily function as vulnerability exploiters, executing attacks in simulators or conducting penetration tests on hosts. *CTF* competitions necessitate employing reconnaissance tactics and techniques before launching an attack to accomplish objectives. Despite experiments involving penetration testing on real hosts, *RL* agents participating in *CTF* scenarios were absent from the reviewed experiments.

Table 2 presents the primary research related to the theme discussed in this article, focusing on experiments involving the automation of cyber attack operations. The table is organized into 6 columns, where: *Env* denotes the type of environment used in the experiment, categorized as either *Simu* for simulations or *Emu* for emulations. Emulation environments are more realistic, incorporating physical hosts, virtual machines, or containers to execute attacks. *Algorithm* specifies the reinforcement learning algorithm employed. *Scope* refers to the variables analyzed in the RL agent experiments, such as steps, rewards, and failures. *Learning Curve Metrics* indicates whether metrics such as offset, speed, and generalization were used to evaluate the experiments. *Dyn* identifies whether the experiments considered dynamic scenarios, and *Ref* specifies the research in which each experiment was conducted.

The analysis methodologies employed in the research summarized in Table 2 primarily focus on steps and rewards. Failures in the kill chain during attacks were not considered, despite some simulators employed, count with this variable in their simulations. Regarding the analysis of key aspects of learning curves, the studies listed in Table 2 lack a detailed investigation of these aspects. In contrast, the research presented in Horta *et al*. [2024b], which is extended in this article, provides an in-depth analysis of *RL* agent attacks under the key aspects of learning curves. Moreover, only Horta *et al*. [2024b] and this extended work considered experiments conducted in dynamic scenarios.

All environments shown in the Table 2 are based on simulators and emulation with hosts. None of the experiments in Table 2, except Horta *et al*. [2024b] and this extended work, employed techniques for reconnaissance, simulated dynamic environments, changing something in the state to analyse the implications, or analysing failures during the attacks. Considering realistic attack scenarios, such as penetration tests or *CTF* competitions, attackers typically lack prior information about the target infrastructure. They must employ specific techniques to evade the security, avoid being detected and keep the progression of the attack. Furthermore, realistic environments are dynamic and changing with exploited vulnerabilities possibly being corrected. Simulators, even those using physical or virtual machines, have difficulties to accurately reflect these real-world complexities. Horta *et al*. [2024b]

Table 2. Related works with experiments.

Env	Algorithm	Scope	Learning Curves Metrics	Dyn.	Ref.
Simu.	NDSPI-DQN dec.	Steps, Rewards	Speed, Generalization	No	Zhou *et al*. [2021]
Simu.	CLAP(PPO+RND)	Steps, Rewards	Speed, Generalization	No	Yang and Liu [2022]
Simu.	HA-DQN	Steps, Rewards	No	No	Tran *et al*. [2021]
Simu.	DQN + LSTM	Steps, Rewards	No	No	Standen *et al*. [2021]
Emu.	A3C/DPPO/GAIL	Steps, Reward, Loss	Speed, Generalization	No	Chen *et al*. [2023]
Emu.	DQN	Steps, Rewards	No	No	Li *et al*. [2022]
Emu.	Random Forest	Rewards	No	No	Holm [2022]
Emu.	KCC (GIWRF)	Steps, Rewards, Failures	Offset, Speed, Generalization	Yes	This work

and this extended work adopt a low-abstraction interface for enabling attack execution and analysis between *RL* agents and the environment, named Exoskeleton. This interface can be integrated into a real *CTF* scenario specifically designed for testing, allowing *RL* agents to perform and analyse attacks in a realistic setting. These types of approaches are important for a better understanding of the behaviour and vulnerabilities of security systems in realistic scenarios. Without these elements, the experiments conducted may have limitations in their ability to faithfully reproduce the conditions encountered in real environments.

4 Methodology

The Methodology Section introduces the *Kill Chain Unscrambler* and its primary component, the *Kill Chain Catalyst*, designed to address the limitations of conventional *RL* algorithms in sequencing tasks within stochastic environments, particularly in *CTF* scenarios. Given that the scenarios employed in the experiments of related works utilize simulators like *NASim* and *CybORG*, which highly abstract real-world environments. This section explores key aspects relevant to the *RL* experiments, including the agent, environment, action space, state space, and the method developed for comparing the metrics of learning curves. These aspects are supported by the *Exoskeleton*[1], a low-abstraction interface developed during this research to perform experiments with *RL* agents using various algorithms more closely to realistic environments.

4.1 Kill Chain Unscrambler

The Kill Chain Unscrambler (*KCU*) is specifically designed to overcome difficulties in identifying attack sequences in attacks, where the target scenario is unknown, as opposed to those performed against simulated or emulated ones. In contrast to other algorithms guided by reward maximization, *KCU* focuses on achieving an optimal sequence in a few steps while minimizing failures.

Unlike traditional *RL* algorithms like *PPO*, *A2C*, *TRPO* and *DQN*, *KCU* takes a slightly unorthodox approach by utilizing recurrent fits in Decision Tree logic instead of neural network refits as its engine. Therefore, *KCU* algorithm for *RL* is implemented through the *Decision Tree Policy*.

The *Decision Tree Policy* implements the *Tree classifier* using *Random Forest*, handling the fitting and prediction processes. This module aims to capture and learn the best patterns in terms of step size, undergoing a *KCU* transformation process from observed data to make predictions without relying on neural networks.

The selection of *Random Forest* within the *Decision Tree Policy* for the *KCU* approach is driven by the problem posed by *RL* agents in attack sequence prediction in a *CTF* environment. In such scenarios, where an attacker navigates an unknown environment with limited exploratory experience, the application of neural networks, which typically require large datasets to perform well, becomes constrained. In contrast, *Random Forest* does not demand extensive datasets for effective model performance, making it suitable for situations where exploratory actions and labelled examples are limited.

Moreover, due to its foundation in decision trees, *Random Forest* enhances the interpretability of the decisions taken by the model. Interpretability is essential in security-related applications where understanding the underlying logic behind an agent's actions contributes to assessing model robustness and providing insights into the attack sequence predictions. This contrasts with neural network architectures, where the decisions taken are less interpretable and more complex to explain. Furthermore, decision tree models require less computational power than training complex neural networks, particularly in deep learning contexts, denoting the computational efficiency of *Random Forest*.

Random Forest model allows for the integration of the *GIWRF*, a variant that applies weighted values to samples during an attack. By assigning higher weights to samples that produce more effective decisions, *GIWRF* provides a means to prioritize influential samples, thereby refining the model's predictions for optimal attack sequences and contributing to an efficient learning process.

The *KCU* class, extending the *BaseAlgorithm* from *Stable Baselines 3*[2], orchestrates the overall functionality of the *KCU* algorithm. It can be configured to use the *DecisionTreePolicy* class as the underlying policy. During training, it collects rollouts, filters the best sequences, and then trains the selected policy accordingly.

Complementary to *KCU*, the *KCC* is a catalyst inspired by bio-informatics, genetic alignment processes, and the use of enzymes to accelerate the decomposition of proteins. Drawing parallels from the biological realm, *KCC* is integrated into the *KCU*. Its primary objective is to accelerate the convergence of the learning curve within *KCU*, seeking sequences with a reduced number of steps, and to minimize failures in

[1] `https://gitlab.com/antonio50/exoskeleton`

[2] `https://stable-baselines3.readthedocs.io`

the chaining of attack techniques to achieve more stealthy and realistic attack sequences.

In this process, genetic alignment has an important role in aligning the kill chains, which act as proteins. The consensus of these sequences aligned by *Needleman-Wunsch* algorithm generates weights for each technique. This weight vector assigns importance to each item in the sequence within a training sample, acting akin to an enzymatic process to accelerate the search for optimal kill chain sequences.

4.2 KCC Agent

Figure 2 presents the operational process of a *CTF* performed by the *KCU Agent* with the catalyst. The flow involves three distinct lanes: the *KCC Agent*, the *Exoskeleton*, and the *Environment*. Initially, the *KCC Agent* starts the exploration cycle due to its lack of prior knowledge about the environment. During this phase, the agent dynamically draws an available and known index of an attack technique from its action space, which refers to the set of all possible actions the agent can execute at a given decision point in the environment, according to its current state and policy. This serves as the initial step into the environment that will be executed through a series of interactions calculated and mediated by the *Exoskeleton*. The *Exoskeleton* acts as an interface that executes actions in the *Environment*, connecting the decisions of the *KCC Agent* and the external system.

In the *Exoskeleton* lane, the index of the selected attack technique undergoes a transformation into a concrete action. This transformation is grounded in a built-in knowledge base derived from the MITRE ATT&CK framework, which maps each technique to a corresponding executable procedure. For instance, the technique T1595.001 (Active Scanning: Scanning IP Blocks) is operationalized through the execution of the real *nmap* tool, directly within the environment. Following the execution of the action in the *Environment*, a set of observables, representing the outcomes of the executed action, is returned. Among the data collected are elements such as service banners retrieved from network scans, IP addresses, and the identification of open ports. These observables encapsulate information about the environment's response, forming the basis for subsequent calculations within the *Exoskeleton*.

The *Exoskeleton* processes the returned observables by computing key metrics, including rewards, failures, and step counts. These metrics are then integrated with the observable data to form a state representation, which comprises observable vectors, raw data, and calculated metrics. This state representation is propagated along two parallel paths that converge once the model has been trained and is ready for prediction tasks. One path feeds the training process of the predictive model, enabling it to learn patterns and associations within the environment. The other path supports the exploitation phase of the *KCC Agent*, guiding its decision-making based on the current knowledge accumulated during exploration. During training, the agent generates a vector mask to isolate states associated with positive rewards. This filtered data is then adjusted by applying a shift of -1 to the action sequence, represented as an action vector. The purpose of this operation is to align each observed state with the action that led to the subsequent state, establishing a correlation between the decision and its resulting outcome. As a result, pairs of state-action vectors are constructed and stored in a buffer for future training iterations.

The training process employs the *Needleman-Wunsch* algorithm as a catalyst for aligning sequences of actions, thereby generating weighted training samples. The weight associated with each sample corresponds to the arithmetic mean calculated from the consensus sequence, which is defined by the most frequent action observed at each aligned position across all sequences. These weights, alongside the stored pairs of state-action vectors, are used as inputs for the *GIWRF* training. The training phase updates the predictive model, refining the selection of attack techniques for future exploitation phases. This iterative process continues until the *KCC Agent* reaches a terminal state, ensuring an adaptive exploration and exploitation cycle for the *CTF*.

The algorithm used to guide the *RL* Agent at time step t is detailed in the following: from the *Exoskeleton*, the *KCU* agent receives a triple $\theta_t = [\overrightarrow{S}_t, \overrightarrow{R}_t, \overrightarrow{A}_t]$ where $\overrightarrow{S_t} = [s_i, s_{i+1}, ..., s_n]_t$, $\overrightarrow{R_t} = [r_i, r_{i+1}, ..., r_n]_t$ and $\overrightarrow{A_t} = [a_i, a_{i+1}, ..., a_n]_t$ are vectors that represent respectively the sequence of *States*, *Rewards* and *Actions* associated to the agent at the end of each epoch t. Then, the agent creates a mask $\overrightarrow{J_t} = [j_i, j_{i+1}, ..., j_n]_t$ based on the indices i of the reward vector $\overrightarrow{R_t}$ that are greater than 0, as stated by Equation 5, indicating positive effects without penalties or failures. $\overrightarrow{J_t}$ serves as a mask to filter the elements from $\overrightarrow{R_t}$, $\overrightarrow{S_t}$ and $\overrightarrow{A_t}$ vectors, which did not produce failures. $\Omega_I(\overrightarrow{X_t})$ is the function to apply the mask, retrieving the values from vector $\overrightarrow{X_t}(\overrightarrow{X_t} \in \{\overrightarrow{R_t}, \overrightarrow{S_t}, \overrightarrow{A_t}\})$ whose indices belong to $I = \{i \in \{1, ..., n_t\} \ / \ j_i = 1\}$ as indicated by Equations 6, 7 and 8.

$$\text{For each } i,\ i \in \{1, ..., n\},\ j_i = \begin{cases} 1, \text{ if } r_i > 0 \\ 0, \text{ if } r_i \leq 0 \end{cases} \tag{5}$$

$$\overrightarrow{R_t} = \Omega_I(\overrightarrow{R_t}) \tag{6}$$

$$\overrightarrow{S_t} = \Omega_I(\overrightarrow{S_t}) \tag{7}$$

$$\overrightarrow{A_t} = \Omega_I(\overrightarrow{A_t}) \tag{8}$$

Then, according to Equation 9, a shift of -1 is applied to the indexes of *Actions* ($\overrightarrow{A_t}$) vector, generating a state-action vector, which is stored in the buffer to be used for training.

$$\overrightarrow{A_t} = \overrightarrow{(A_t)}_{i-1} \tag{9}$$

Next, the catalysis is initiated by applying the *Needleman-Wunsch* algorithm $NW(\overrightarrow{A}_t, QA) = K$ [Gancheva and Stoev, 2023], where $K = \{\overrightarrow{K}_c, \overrightarrow{K}_{c+1}, ..., \overrightarrow{K}_m\}$ is a set of aligned vectors from c up to m elements. $NW(\overrightarrow{A}_t, QA)$ takes as input the *Actions* vector $\overrightarrow{A_t}$ and the set of vectors stored in the buffer *Best KC QUEUE* (*QA*), deriving the consensus vector $\overrightarrow{KCC_t}$ through the mode, as shown in Equation 10.

$$\overrightarrow{KCC_t} = \text{mode}(NW(\overrightarrow{A}_t, QA)) \tag{10}$$

Then, the catalysis process extracts the weight vector $\overrightarrow{W_t} = [w_i, w_{i+1}, ..., w_n]_t$, calculated by normalizing each element i from consensus vector $\overrightarrow{KCC_t}$ with the sum of the value in position z for each element $\overrightarrow{K} = [k_z, k_{z+1}, ..., k_n]$ across the m aligned vectors in K, as described in Equation 11.

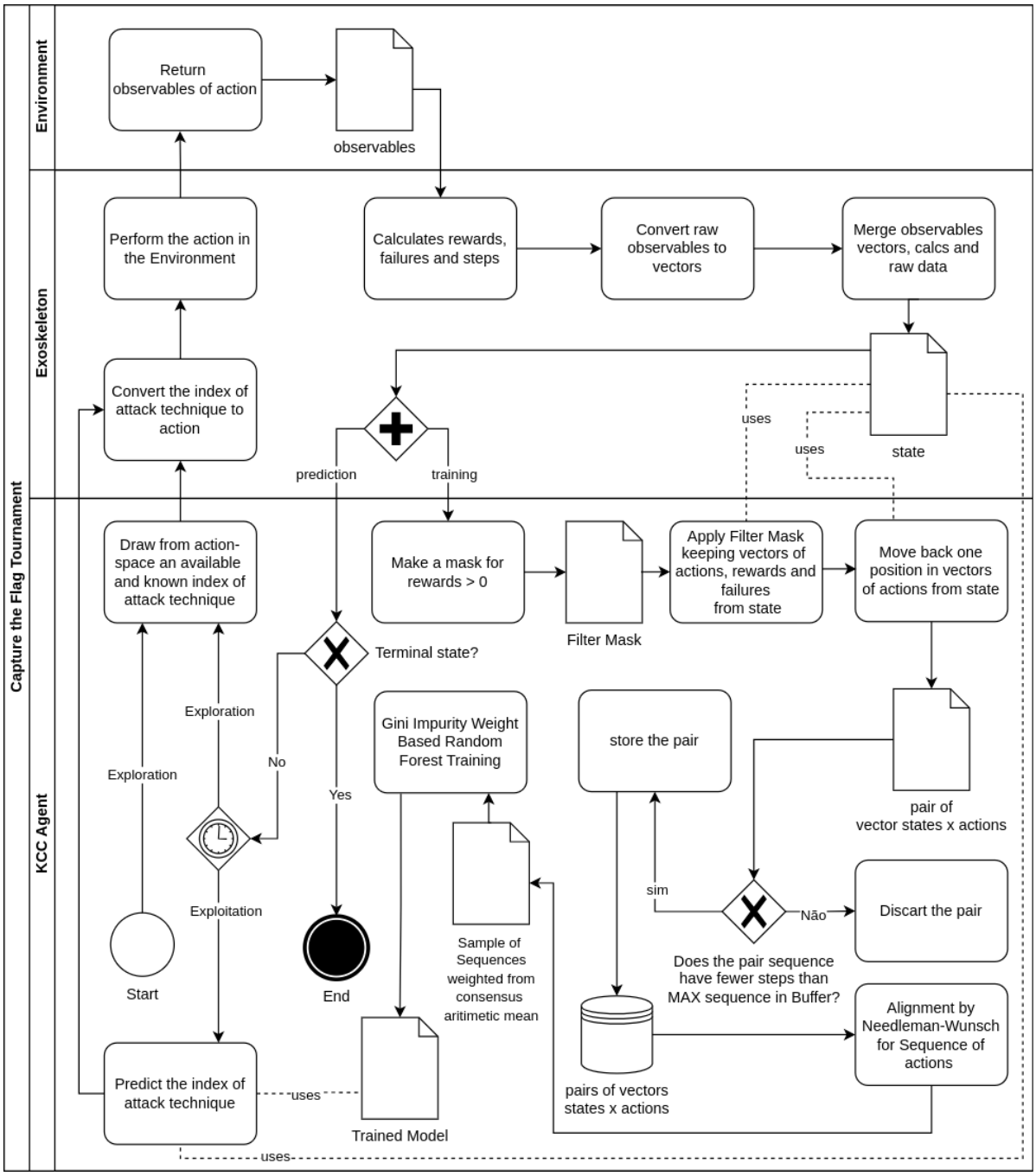


Figure 2. The Kill Chain Unscrambler with Catalyst Agent.

$$\text{For each } i,\ i \in \{1, ..., n\},\ w_i = \frac{(\overrightarrow{KCC}_t)_i}{\sum_{z=1}^{m} (k_z)_i} \qquad (11)$$

At this stage, *Rewards* $\overrightarrow{R_t}$, *States* $\overrightarrow{S_t}$, and *Weights* $\overrightarrow{W_t}$ enter the rollout process. Rollout involves determining actions based on the reinforcement learning training policy, employing a decision tree policy based on *GIWRF* (Equation 4). This policy formation uses the weight vector for each sample. If the resulting action sequence $\overrightarrow{B} = max(QA)$, where *max(QA)* is a function that retrieves the vector with the highest dimensionality in the set *QA*, which is inserted into the queue in descending order based on the dimensionality (Equation 12).

$$\text{if } \dim(\overrightarrow{A}_t) < \dim(\overrightarrow{B}) \text{ then } QA = QA \cup \{\overrightarrow{A_t}\} \qquad (12)$$

The randomized or predicted action is sent to the scenario, initiating a feedback loop until all epochs are processed producing the *Decision Tree Policy*: $\pi_\theta(S_t|A_t, W_t)$.

The *KCU* Agent has different operational modes that can be activated through hyper-parameters based on the objectives to be achieved. These modes include: *KCU only*, *KCC*, and *Dynamic*. In the *KCU only mode*, the algorithm follows the procedures described above to find attack sequences without the use of the catalyst. Additionally, the *KCC mode* activates the Catalyst to optimize the sequence discovery process using weights derived from the arithmetic mean consensus generated by *Needleman-Wunsch* alignment. Finally, the *Dynamic*

mode monitors the agent's training, and if the last step is equal to the maximum steps per epoch and if the last step exceeds the standard deviation from moving average of steps for the last 10 episodes, the buffer (Best KC QUEUE) is reset to zero, and the epsilon (e-greedy) value is reset to the initial state. This process encourages the agent to discover new attack strategies.

4.3 Environment

The environment constitutes the element responsible for presenting a state to which the agent responds by selecting an action, subsequently receiving a reward or punishment that results in a new state. This interaction forms the basis of the learning process, as the agent progressively adapts its behaviour based on feedback. In the context of this study, the environment is represented by a *CTF* scenario, where state transitions, actions, and rewards are defined by the dynamics and objectives inherent to the *CTF* setting.

In this context, a server has been configured with the characteristics of a *CTF* challenge to establish the *Dummy* scenario for attack practice, using images from *vunlhub*[3]. This server operates within a container that runs on the Ubuntu Linux operating system. It contains two text files, each housing secret keys symbolizing flags in the *CTF* scenario, the ultimate objectives. One of these files resides in the home directory of the basic user, while the other is located in the root directory, necessitating privileged access for content retrieval. Ensuring a secure testing environment, the server has all applicable security patches applied and is devoid of any known vulnerabilities.

[3] `https://github.com/vulhub/vulhub/tree/master/libssh/CVE-2018-10933`

Regarding integration with *RL*, an interface named *Exoskeleton* has been devised to facilitate autonomous attack processes. This interface operates at a low level of abstraction, allowing reinforcement learning algorithms to interact seamlessly with the target server. Within the *Exoskeleton*, the agent is equipped with 9 *MITRE ATT&CK*[4] techniques that can be executed against the server. These techniques leverage known attack tools and generate responses based on the success or failure of potential actions within the scenario.

4.3.1 Action Space

Table 3 presents a set of *MITRE ATT&CK* techniques implemented in the *Exoskeleton* interface. Each row details a specific tactic and technique combination, providing information such as tactic and technique IDs, names, and associated tools. These techniques serve as the action space for autonomous attack simulations, allowing for the evaluation of reinforcement learning algorithms in a controlled and realistic environment. Each of the tools correlated with these techniques was implemented as commands, as described in Table 4, to be executed against the *CTF* environment. The complete references and detailed descriptions of these techniques can be accessed on the *MITRE ATT&CK*.

Based on the outlined techniques, the shortest kill chain available to exploit the *dummy* scenario in the *Exoskeleton* is depicted in Figure 3. Kill chain A (KC A) represents a scenario with updated security patches characterized by a minimal number of steps and failures. It includes reconnaissance activities such as scanning IP blocks and word list scanning, along with gathering victim identity information, involving the acquisition of email addresses. Further techniques encompass credential access through brute force attacks, such as

[4] `https://attack.mitre.org/matrices/enterprise/`

Table 3. Exoskeleton's action space.

Tactic id	Tactic name	Technique id	Technique name	Tool
TA0043	Reconnaissance	T1595.001	Active Scanning: Scanning IP Blocks	nmap
TA0043	Reconnaissance	T1595.003	Active Scanning: Wordlist Scanning	dirb
TA0043	Reconnaissance	T1589.002	Gather Victim Identity Information: Email	script parser
TA0001	Initial Access	T1078.003	Valid Accounts: Local Accounts	hydra
TA0001	Initial Access	T1190	Exploit Public-Facing Application	libssh exploit
TA0006	Credential Access	T1110.001	Brute Force: Password Guessing	ssh
TA0007	Discovery	T1083	File and Directory Discovery	bash
TA0004	Privilege Escalation	T1548.001	Abuse Elevation Ctrl Mechanism: Setuid & Setgid	find
TA0009	Collection	T1005	Data from Local System	cat

Table 4. Exoskeleton's commands by techniques and tools.

Technique id	Tool	Command
T1595.001	nmap	nmap -sV {dst_ip_address}
T1595.003	dirb	dirb {service_name}://{dst_ip_address}:{portid} -w {wordlist} -S
T1589.002	script parser	re.compile(r'\b[A-Za-z0-9._%-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b', re.IGNORECASE)
T1078.003	hydra	hydra -l {username} -P {password_list} {service_name}://{dst_ip_address}:{portid}
T1190	libssh exploit	`https://www.exploit-db.com/exploits/46307`
T1110.001	ssh	ssh {username}@{dst_ip_address}
T1083	bash	find / -perm -u=s -type f 2>/dev/null
T1548.001	find	find / -name root.txt -exec cat {} 2>/dev/null
T1005	cat	cat /root/root.txt

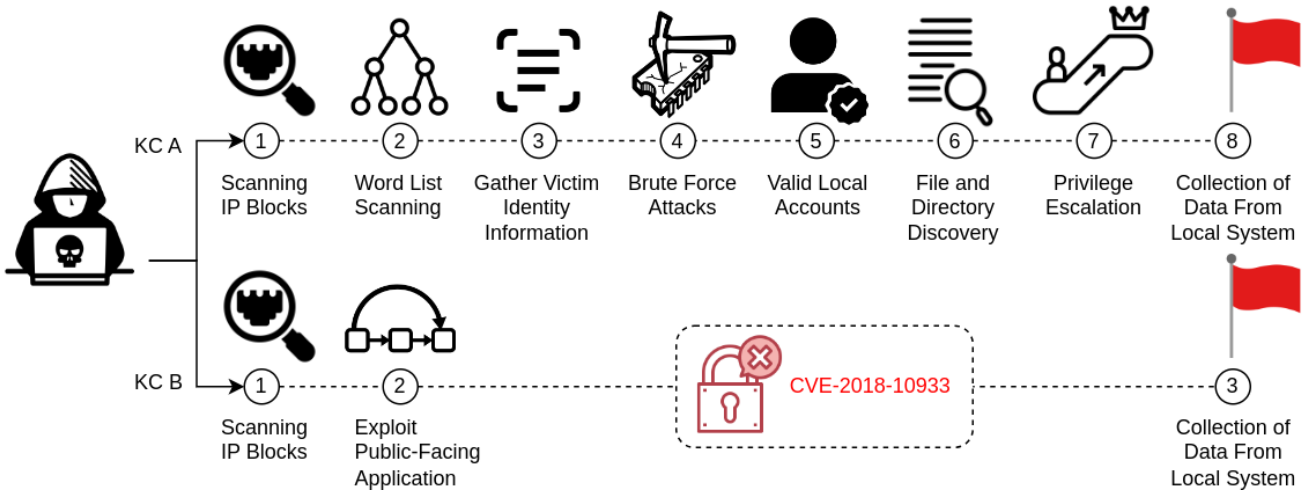


Figure 3. Kill chains available in Exoskeleton's Dummy Scenario.

password guessing for SSH, and initial access through valid local accounts for SSH. Tactic discovery manifests through file and directory discovery in SSH, leading to the exploitation of privilege escalation via the *Setuid* and *Setgid* elevation control mechanisms in SSH. Ultimately, data collection from the local system culminates in the achievement of the goal of capturing the flag. In contrast, Kill chain B (KC B), illustrated in Figure 3, serves as an alternative kill chain when the target machine is not patched. This scenario exploits a vulnerability in *libssh*, enabling the attacker to bypass authentication control and gain a session with root privileges, as explained in *CVE-2018-10933*[5].

Exoskeleton transforms real computing systems into a *Gymnasium*-compatible[6] scenario format, a platform tailored for testing reinforcement learning algorithms.

Integral to the reinforcement learning process, the *Exoskeleton* adopts a reward model. Successful actions resulting in state modifications are rewarded with a +1, actions that execute without altering the state receive a reward of 0 (e.g., repeated commands), and failures to execute, whether due to connection issues or service failures, result in a penalty of -1. Therefore, achieving both flags within the scenario leads to a substantial reward of 100.

4.3.2 State Space

The *Exoskeleton* conceptualizes the *State Space* by encapsulating the outcomes derived from the agent's executed actions in their raw format, such as the output of an *nmap* command as it manifests in *stdout*. The *State* fields, representing various facets of the system, are enumerated in the Table 5. Table 5 organizes the fields into two columns: the first delineates the field, and the second furnishes a succinct description. Moreover, to ensure compatibility with *RL* algorithms in a *POMDP* scenario, as in the case of *CTF*, *Exoskeleton* transforms the *State* into a vector of integers representing *Observables*, structured according to Table 6.

The invariant model proposed by Janisch *et al*. [2023], which processes each host feature vector individually with a shared embedding function, served as the basis for the modelling of *Exoskeleton*. According Janisch *et al*. [2023], the invariant architecture can process an unlimited number of hosts and is better equipped for generalization due to weight sharing, while using a fraction of the parameters. Therefore, the *Exoskeleton* transforms the *State* into a vector of integers representing *Observables*. The modelling of *States* and *Observables* allows a large scalability for *States*. This modelling become *Actions* techniques independent of the variables in the scenario. The independence generated by this modelling is an important factor in distinguishing *Exoskeleton* from

[5] `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-10933`

[6] `https://github.com/Farama-Foundation/Gymnasium`

Table 5. Exoskeleton's state space.

Field	Description	Field	Description
src_ip_address	IP address of the attacker machine	session_used	ID of the used session
dst_ip_address	IP address of the target system	session	List of available sessions
action_executed	Action executed to attain this state	step	Step received for executing the action
results	Results of the action	done	Indication of the objective achieved
tactic	Tactic of the executed action	recon	IPs and services gathered by nmap
technique	Technique of the executed action	credentials	IPs and services gathered by hydra
failed	Indication if the action failed	discovery	Discovery data: login, password, vulns
reward	Reward from the executed action	collection	List of collected data, flags, etc.

Table 6. Exoskeleton's observables as a vector of integers.

Field	Description	Field	Description
dst_addr	Representation of the targeted IP	session	Representing the used session
dst_addrs	Quantity of targeted IPs	sessions	Denoting the available sessions
act_exec	Representation of executed action	pv_sessions	Indicating the amount of privileged sessions
actions	Representing executed actions	recons	Amount of data obtained in recon
tactic	Representation of executed tactic	credentials	Amount of captured credentials
tactics	Denoting the available tactics	valid_cred	Representing the number of valid credentials
tech	Representation of the technique	discovery	Denoting the amount of discovered data
techs	Quantity of available techniques	collections	Indicating the amount of collected data
failed	Indicating the action failed	step	The step at the action was executed
reward	Representation of the reward	done	Indicating whether the objective was achieved

simulators employed in related work.

4.4 Metrics for Learning Analysis

In *RL*, agents initiate the training without prior knowledge of the environment, requiring exploration to optimize their behaviour over time. The *offset* metric represents the agent's initial performance level, captured at the beginning of training. This metric establishes the baseline performance and reflects the agent's ability to interact with the environment before acquiring significant knowledge. For *RL* tasks, the *offset* illustrates the agent's capacity to explore states and take actions in the absence of environmental familiarity. In this paper, the *offset* is determined by the arithmetic means of the criterion $c_{initial}$ for reward, steps or failures in the first epochs for each series per experiment.

As training progresses, the *speed* metric quantifies the rate at which the agent improves its efficiency by reducing failures, steps and increasing rewards. *Speed* is evaluated as the time required to reach a threshold of the asymptotic performance, which is directly influenced by the training data, the structure of the reward function, and the model's complexity. This metric highlights the efficiency of the learning process, as faster agents demonstrate an ability to identify and exploit patterns in the environment more effectively. For reinforcement learning, *speed* measures how quickly the agent adapts its strategy to increase rewards.

Moreover, assessing the learning *speed* requires identifying the stage at which the learning rate reaches an asymptotic level of performance. In this paper, the term *generalization point* e_{gp} refers to when the asymptotic performance stage is achieved, determined when the combination of both mean and standard deviation conditions is satisfied.

Therefore, the e_{gp} is considered reached at the first episode e such that, for the subsequent persistence over k episodes it fulfils these 2 conditions. The first condition considered is when the standard deviation $\sigma_e < \tau$ of the number of criterion c for rewards, steps, or failures within a window of size w is less than a threshold τ. Where σ_e is the standard deviation calculated as Equation 13 and the desired dispersion limit τ, defined for each type of problem:

$$\sigma_e = \sqrt{\frac{1}{w}\sum_{i=e-w+1}^{e}(c_i - \bar{c}_e)^2} \tag{13}$$

and $\bar{c}_e$ is the expected value of rewards, steps, or failures in the same window, as shown in Equation 14:

$$\bar{c}_e = \frac{1}{w}\sum_{i=e-w+1}^{e} c_i \tag{14}$$

The second condition is $\bar{c}_e < M$, the mean of the number of c for rewards, steps, or failures in the same window is less than a maximum acceptable threshold M.

Subsequently, learning *speed* can be determined, considering the Equation 15 for learning curves exhibiting a *generalization point*:

$$\textit{Speed} = \frac{|c_{gp} - c_{initial}|}{e_{gp} - e_{initial}} \tag{15}$$

Where $c_{initial}$ represents the mean of rewards, steps, or failures at the first episodes from experiments. c_{gp} denotes the mean of rewards, steps, or failures at the *generalization point*. $e_{initial}$ represents the first episode number and e_{gp} is the episode number where the *generalization point* was reached.

For curves lacking a *generalization point*, the mean of the last episode e_{last} and the criterion c_{last} of the experiments will be considered to estimate the *speed*, as shown in Equation 16.

$$\textit{Estimated Speed} = \frac{|c_{last} - c_{initial}|}{e_{last} - e_{initial}} \tag{16}$$

The final phase of training is characterized by the agent achieving *asymptotic performance* stage, which is captured by the *generalization* metric. *Generalization* indicates the agent's ability to consistently make good decisions across a wide range of scenarios within the environment. This stage signifies that the agent has successfully learned the environment's dynamics, pathways, and optimal strategies to achieve its goals. In reinforcement learning, *generalization* reflects the agent's capability to maximize rewards while minimizing failures and unnecessary steps, showcasing the convergence of the learning process to a stable and effective policy. For the experiments carried out, the *generalization*, presented in Equation 17, was determined by mean of the criterion achieved from the *generalization point* up to the experiment's conclusion.

$$Generalization = \frac{1}{e_{last} - e_{gp}} \sum_{i=e_{gp}}^{e_{last}} c_i \qquad (17)$$

5 Experiments

The experiment involving *KCC* will be assessed alongside four algorithms: the traditional value-based *DQN*, policy-based *PPO*, *TRPO*, and the hybrid *A2C RL* algorithms. While these algorithms use neural networks as their policy, *KCC* employs decision tree logic. The assessment will focus on key metrics of the learning curve, in relation to total rewards, total steps, and total failures.

To analyse the performance of *KCC* in attack operations, considering the learning curve for limited attack experiences and ability to handle dynamic scenarios, the experiment is structured into two independent essays. The first essay aims to compare the learning curve of the *KCC*, with four other *RL* algorithms: *A2C, DQN, PPO*, and *TRPO*. The last essay demonstrates the resilience of the *KCC* to dynamic scenarios with the *dynamic_scenario* flag activated, alongside the same four *RL* algorithms aforementioned. In this final essay, vulnerabilities are patched midway through each algorithm's learning cycle, prompting the converged algorithms to explore new attack paths.

For each essay in the experiment, involve conducting 15 attacks using reinforcement learning agents for each of the algorithms in a *CTF* competition. In these essays, the agent has a limited maximum steps per epoch and total epochs to learn, set at 50 for both. The selected environment for this competition is the *Dummy* scenario of the *Exoskeleton* interface. In the last essay, the *Dummy* scenario introduces an additional vulnerability, *CVE-2018-10933*[5], which, as explained, will be addressed during the learning cycle.

The analysis of the agents' performance in unfamiliar scenarios focuses on the number of steps required to achieve the goal, the quantity of received rewards, and the occurrences of failures. For all essays, the parameters outlined in Table 7 were considered. Parameters not explicitly provided will follow the default values of the *Stable Baselines 3 framework*.

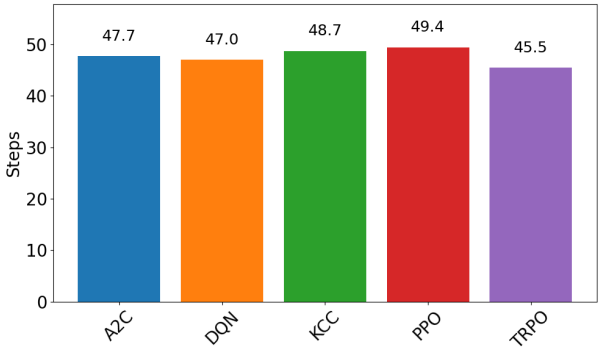
Table 7. Essays' parameters.

Agent	Parameter	Essay 1	Essay 2
All	epochs	50	25 + 25
All	max steps per epoch	50	50
All	timesteps	2500	2500
KCC	epsilon (e-greedy)	0.25	0.50
KCC	decay_rate	0.01	0.02
KCC	kcc (catalyst)	True	True
KCC	seed (random seed)	1 to 15	1 to 15
KCC	buffer	20	10
KCC	n_estimators	50	50
KCC	min_samples_leaf	1	1
KCC	dynamic_scenario	False	True

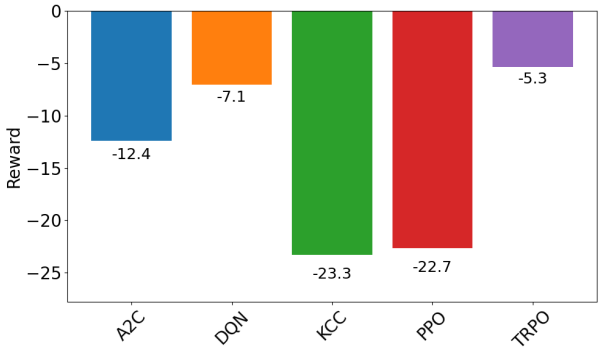
6 Results and Discussion

For the first essay, *KCC* was compared with 4 traditional *RL* algorithms in the *CTF* scenario. As previously mentioned, in *CTF* situations, all algorithms were subjected to a scenario with limited steps and epochs for learning.

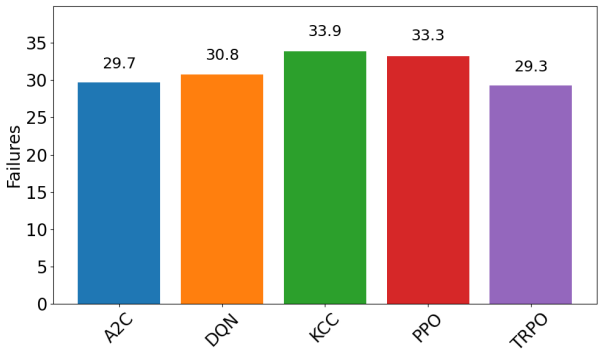
Figure 4 presents the *offset* metric results in the first epoch of training for the *KCC*, *PPO*, *TRPO*, *A2C*, and *DQN* agents. In Figure 4.a, the final step achieved by each agent to complete the *CTF* objective is depicted. The training essay established a maximum limit of 50 steps per epoch. As shown, the average number of steps of the first episodes for each of the agents was very close to the limit of 50. *TRPO* was the one that presented the lowest average number of steps to find the flag, around 45 steps. However, this value is not representative, given that it is also a high value, natural at this stage of the attack, when the agents are completely unaware of the target scenario.



(a) First Episode Steps



(b) First Episode Rewards



(c) First Episode Fails

Figure 4. Comparing the offset among agents.

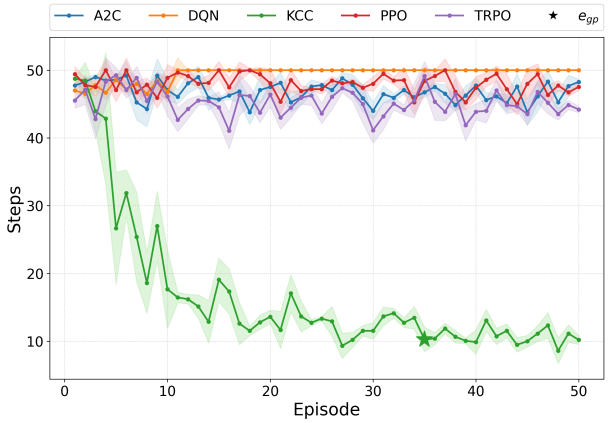
This result is also reflected in Figure 4.b, which illustrates the rewards obtained by the agents. Due to the inability of most agents to complete the objectives within the allowed time, they accumulated more penalties than rewards, resulting in negative reward values.

Figure 4.c shows the failures for each agent, where the *TRPO* agent committed the fewest failures, with 29.3 recorded fails average, followed by *A2C*, *DQN*, *PPO*, and *KCC*. Since the first epoch represents the exploration phase, where agents lack knowledge about the environment, these results reflect the consequences of their exploratory and random actions. Neural network-based algorithms, such as *A2C*, *PPO*, *DQN*, and *TRPO*, benefit from the ability to adjust synaptic weights using gradient descent during training in each episode, enabling some degree of learning even within the initial epoch. Conversely, the *KCC* agent, which employs a decision tree policy, begins constructing its predictive model only in subsequent epochs, after accumulating sufficient experience from prior interactions. Regarding *offset*, the inferior performance of *KCC* compared to the other agents is justified by the nature of decision tree algorithms. Unlike models that allow weight updates, decision trees are always built from scratch at the end of each episode, preventing incremental learning.

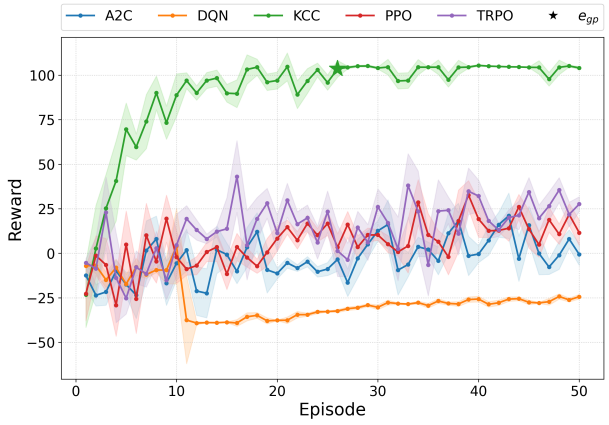
Considering the *speed* and *generalization* metrics, the learning curves for steps, rewards, and failures are presented in Figure 5. As shown in Figures 5.a, 5.b and 5.c only *KCC* generalized, achieving an average of 10.71 steps kill chain. This serves as evidence that other algorithms require more experience to learn and converge. This essay demonstrates the issues faced by *RL* algorithms based on neural networks when dealing with limited data for training, a factor present in the sequencing of real attack situations.

Table 8 presents a comparison of the learning curves obtained from the experiment across the metrics *offset*, *speed*, and *generalization*. For the evaluation of the learning curves, the parameters w and k were both set to 3. The τ thresholds were defined as 0.5 for failures, and 0.75 for both rewards and steps. Values close to those achieved by *KCC* were used as a reference for setting the threshold M, with values of 90 for rewards, 17 for steps, and 3 for failures. The best results are highlighted in bold.

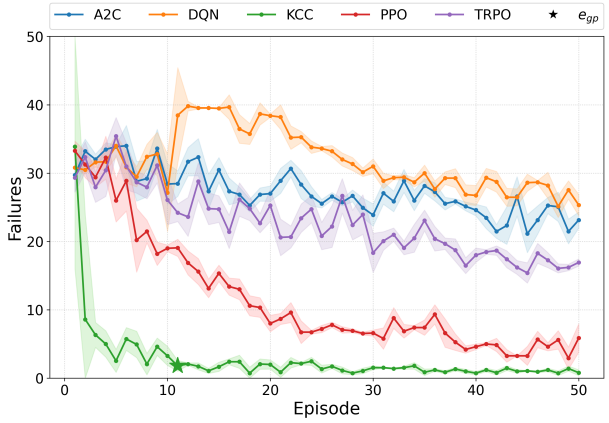
In the *offset* evaluation shown in Table 8, *TRPO* outperformed the other algorithms, achieving the best values for reward, steps, and failures, indicating a more efficient learning process during this phase. For the *speed* metric, which reflects the agents' ability to adapt and improve performance, *KCC* achieved the highest scores across all categories: 5.07 for reward, 1.13 for steps, and 3.21 for failures, surpassing



(a) Last Step x Episode



(b) Reward x Episode



(c) Failures x Episode

Figure 5. The learning curve for agents with limited experiences.

Table 8. Learning Curve Analysis.

Series	Offset			Speed			Generalization		
	Reward	Steps	Failures	Reward	Steps	Failures	Reward	Steps	Failures
A2C	-12.40	47.73	29.73	0.24	-0.01	0.13	-	-	-
DQN	-7.07	47.00	30.80	-0.35	-0.06	0.11	-	-	-
KCC	-23.27	48.73	33.87	**5.07**	**1.13**	**3.21**	**103.39**	**10.71**	**1.40**
PPO	-22.67	49.40	33.27	0.70	0.04	0.56	-	-	-
TRPO	**-5.33**	**45.53**	**29.33**	0.67	0.03	0.25	-	-	-

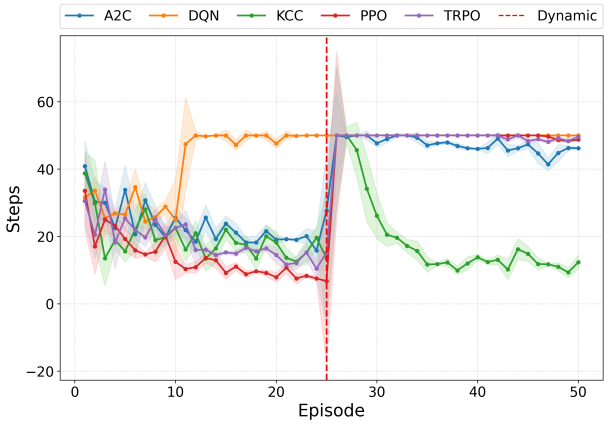
the other algorithms. This result highlights the advantage of decision tree-based policies in adjusting to the task dynamics during the learning process. Finally, in the *generalization* phase, none of the other algorithms reached acceptable performance levels, where only *KCC* was able to meet the defined performance thresholds, with values of 103.39 for reward, 10.71 for steps, and 1.40 for failures.

In addition, Table 9 presents the percentage differences in the average results per episode for each experiment, comparing them to the outcomes achieved by *KCC*. Leveraging decision tree logic, in contrast to the neural networks used by the other algorithms, *KCC* demonstrated superior overall performance for this type of problem. On average, *KCC* required 16.56 steps to reach the objective by the end of the learning cycle, while the other algorithms averaged around 47 steps. A similar pattern was observed in the rewards metric, where *KCC* achieved an average of 90.74, outperforming the second best result from *TRPO* with 13.83, and far surpassing the worst result from *DQN*, which recorded -26.70. Furthermore, when analysing the average number of failures per episode, the differences among the algorithms became even more evident. *KCC* once again stood out, achieving the lowest average of 2.66 failures per episode, followed by *PPO*, *TRPO*, *A2C*, and *DQN*, with the latter exhibiting a difference of over 1000% compared to *KCC*.

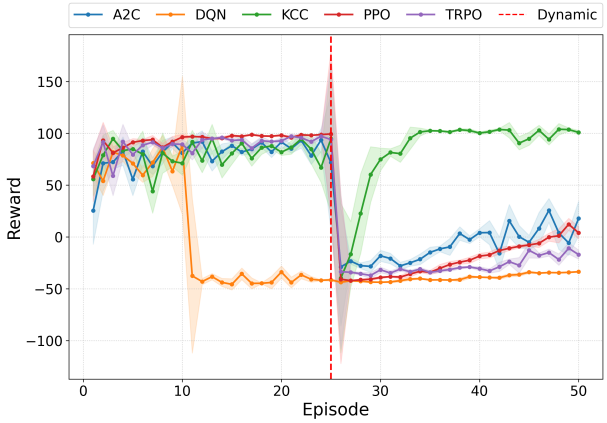
The second essay, which commenced with the exploitation potential of CVE-2018-10933 within the *Dummy Scenario*, presents distinctive observations. This vulnerability facilitated the extraction of *KC B*, as delineated in Figure 3. However, during the ongoing learning cycle at step 25, a remediation ensued, restricting access solely to *KC A*, also depicted in Figure 3. This deliberate protocol aimed at scrutinizing agent behaviour concerning resilience in dynamic scenarios.

Figure 6 encapsulates the comprehensive evaluation of agents subjected to dynamic scenarios. The red vertical demarcation (dynamic) signifies the vulnerability-fixing juncture.

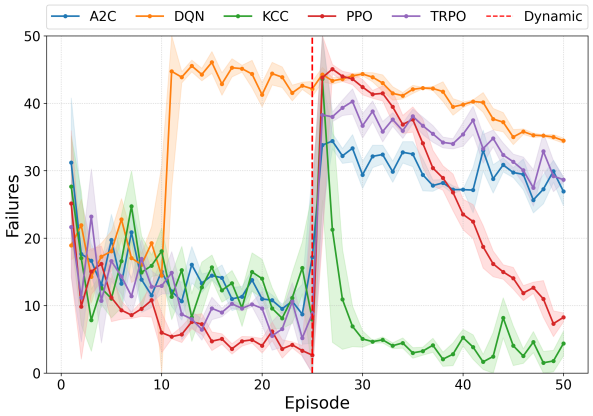
In the second essay, the *KCC* agent was configured to operate in dynamic scenarios, adapting its behaviour based on the observed variations in the outcomes of its actions. When a change in the environment negatively impacted its learning speed or delayed convergence, the agent dynamically recalculated its strategies to explore new paths toward achieving its objectives. The primary focus of this essay was to analyse the offset and learning speed following the correction of a vulnerability during episode 25, which occurred mid-attack. This behaviour is illustrated in Figure 6, where, for all 3 evaluation criteria reward, steps, and failures, *KCC* demonstrated superior performance in the end of attack. After the correction, *KCC* promptly recalculated its trajectory and resumed its progression at a faster speed than the other agents, seeking to achieve its goal and generalize through a new path.



(a) Last Step x Episode



(b) Reward x Episode



(c) Failures x Episode

Figure 6. The learning curve for agents in a dynamic scenario.

Table 10 presents the detailed results for the agents based

Table 9. Percentage Differences Relative to KCC by Average of Episodes per Experiments.

Serie	Steps	Diff Steps	Rewards	Diff Rewards	Failures	Diff Failures
KCC	**16.56**	**0.00%**	**90.74**	**0.00%**	**2.66**	**0.00%**
A2C	46.82	182.66%	-2.72	-103.00%	27.30	926.37%
DQN	49.47	198.69%	-26.70	-129.43%	31.82	1096.39%
PPO	48.02	189.93%	6.02	-93.37%	11.59	335.89%
TRPO	45.23	173.10%	13.86	-84.72%	22.76	755.74%

on offset, speed, and generalization metrics after episode 25. Although *A2C* achieved the best results for offset in episode 26, the values remained high across all algorithms, indicating that previous strategies were no longer effective and that new exploratory efforts were required. In terms of learning speed, *KCC* exhibited an advantage, evidencing its capacity to adapt to dynamic environments by quickly resuming its search for generalization. Regarding generalization itself, none of the algorithms fully generalized after the vulnerability correction; however, the learning trajectories of *KCC*, followed by *PPO*, suggested a tendency towards achieving generalization in subsequent episodes.

Table 10. Learning Curve Analysis after Episode 25.

Series	Reward	Steps	Failures
Offset			
A2C	**-28.93**	**50.00**	**33.80**
DQN	-43.67	**50.00**	44.27
KCC	-39.80	**50.00**	43.60
PPO	-40.80	**50.00**	43.80
TRPO	-33.60	**50.00**	38.27
Speed			
A2C	1.95	0.16	0.29
DQN	0.42	0.00	0.41
KCC	**5.86**	**1.57**	**1.63**
PPO	1.87	0.05	1.48
TRPO	0.69	0.01	0.40

Considering the complete attack cycle in a dynamic scenario, Table 11 highlights the advantages achieved by *KCC* over the others by average of episodes per experiments, with differences reaching up to 138.88% in steps, 121.39% in rewards, and 262.28% in failures.

Table 11. Percentage Differences Relative to KCC.

Serie	Steps	Diff Steps
KCC	**19.04**	**0.00%**
A2C	35.43	86.07%
DQN	45.49	138.88%
PPO	31.73	66.63%
TRPO	34.15	79.34%
	Rewards	Diff Rewards
KCC	**81.57**	**0.00%**
A2C	35.74	-56.18%
DQN	-17.45	-121.39%
PPO	35.58	-56.38%
TRPO	30.55	-62.54%
	Failures	Diff Failures
KCC	**10.16**	**0.00%**
A2C	22.18	118.30%
DQN	36.81	262.28%
PPO	17.84	75.53%
TRPO	23.13	127.67%

7 Final Considerations

With technological advancements, tasks traditionally performed by humans have increasingly been delegated to machines or agents equipped with artificial intelligence, particularly in the field of cybersecurity. In the context of autonomous cyber attacks, *RL* stands out as a widely employed approach. However, *RL*-based methods face significant issues, including limited availability of training data, low adaptability to dynamic environments, and restricted interpretability of decision-making policies. Moreover, in real-world attack contexts, beyond reducing the time-consuming costs associated with red team specialists and agent training, minimizing the number of failures and steps during cyber attacks is critical to lowering the risk of exposure. Horta *et al*. [2024b] introduced the *KCC* to address limitations in the use of *RL* for cyber attacks in the state of the art. The *KCC* incorporates a catalyst inspired by genetic alignment to optimize the search for effective attack chain sequences in which combined with a decision tree-based logic, outperformed all other algorithms in *RL*-driven attack scenarios.

This work builds upon the findings of the original article [Horta *et al*., 2024b], aiming to refine the previous analysis by addressing gaps related to the evaluation of learning and the improvement of experiments. This extension investigates the learning process of agents within the same context as the original study, employing the *KCC* algorithm to minimize failures and steps using limited data, while enhancing resilience in dynamic environments. The focus is on a more detailed analysis of attack sequencing learning, with an emphasis on its applicability in *CTF* scenarios, based on the execution of a larger set of experiments.

Experiments involving more than 150 attack emulations were conducted in the context of an autonomous *Capture the Flag* tournament. These experiments evaluated the learning performance of *RL* agents through metrics derived from their learning curves, including *offset*, *speed*, and *generalization*. This evaluation was based on the metrics equations developed to analyse the learning curves. The analysis incorporated the number of *steps*, *rewards*, and *failures* recorded during the training of the agents. The experiment was divided into two distinct essays. The first essay focused on examining the learning curve of agents in an unfamiliar scenario with limited experiences within the *CTF* tournament. The second essay assessed the resilience of the agents in a dynamic environment, where a vulnerability was patched in the middle of training to evaluate their ability to recalibrate strategies in response to environmental changes.

The results highlighted the learning speed of *KCC* in generalizing and optimizing attack strategies, achieving objectives with fewer steps and failures compared to other algorithms. Differences were observed of up to 198.69% in steps, 129.43% in rewards, and 1096.39% in failures when comparing the performance of *KCC* with traditional *RL* algorithms. Additionally, the ability of *KCC* to adapt to environmental modifications, such as the remediation of vulnerabilities, showcased its resilience in dynamic scenarios. This adaptability distinguished *KCC* from other algorithms tested in the study. These findings emphasize the relevance of decision tree-based methodologies and the potential role of the catalyst

in enhancing the performance of *RL* agents in cyber attack contexts.

However, the offset metric represents a characteristic of the exploration phase inherent to all *RL* algorithms, as it reflects the stochastic nature of initial actions taken by agents in unknown environments. This stochasticity is a drawback for *RL* agents, as it often results in variable performance during the early stages of training. Consequently, among the three learning curve metrics, the offset remains the most sensitive to the randomness of exploratory actions. While the *KCC* algorithm demonstrated advantages in learning for the later phases of training, the offset remains an area requiring further refinement through targeted research. Addressing this limitation could enable *KCC* to gain a more consistent advantage over other *RL* algorithms during the initial stages of training.

This study extends the original contributions related to the *KCC* algorithm and the *Exoskeleton* interface within the context of *RL* for cyber attacks. In addition to these developments, it introduces a method for analysing learning curves based on objective metrics, namely offset, speed, and generalization. This approach enables a more rigorous and less subjective comparison than graphical interpretations. Furthermore, this work expands the number of experiments and provides all results and source code, facilitating the replication of experiments and supporting the development of new studies based on this framework.

Future investigations should focus on enhancing the exploration phase of the *KCC* algorithm, with the aim of mitigating the impact of stochasticity and improving its offset performance. Such improvements may include advanced exploration strategies or hybrid methodologies to balance exploration and exploitation more effectively. Furthermore, experiments in increasingly complex and variable environments would provide valuable insights into the robustness and adaptability of *KCC*, particularly in dynamic cyber attack scenarios.

Declarations

Authors' Contributions

The study was conceptualized collaboratively, with AH, RG, and AS jointly defining the research objectives and establishing the primary directions. AH served as the principal author and took the lead in designing the Kill Chain Catalyst (KCC) algorithm and outlining the experimental framework. AS contributed significantly to the initial brainstorming sessions, particularly by introducing the Genomic Sequencing concept, which shaped the study's foundation. RG provided critical input on the AI approaches and offered continuous feedback throughout the research. AH spearheaded the data analysis and comparative evaluation of the algorithms, supported by discussions with RG and AS. The manuscript's initial draft was written by AH, while RG and AS routinely refined the text, enhancing its fluency and coherence. AH prepared the visual representations and figures to illustrate the findings. RG and AS also supervised the research process, offering guidance and oversight at every stage. All authors reviewed and approved the final manuscript.

Acknowledgements

The authors state that all content within this article, encompassing the research design, experimental execution, data analysis, and the formulation of conclusions, was entirely conceived, developed, and analysed by the authors. This ensures the complete originality of the work presented. To meet the high-quality standards expected for publication in English-language journals and to adhere to the LaTeX format, the Writefull for Overleaf language tool was utilized for spelling and grammar checks. This measure was taken solely to ensure closer alignment with correct English usage and does not reflect any input from AI-based content generation tools.

Competing interests

The authors declare that they have no competing interests.

Availability of data and materials

The datasets and software generated and analysed during the current study are available in `https://gitlab.com/antonio50/exoskeleton`.

References

Al-Azzawi, M., Doan, D., Sipola, T., Hautamäki, J., and Kokkonen, T. (2024). Artificial intelligence cyberattacks in red teaming: A scoping review. In *World Conference on Information Systems and Technologies*, pages 129–138. Springer. DOI: 10.1007/978-3-031-60215-$3_1$3.

Aspland, E., Harper, P. R., Gartner, D., Webb, P., and Barrett-Lee, P. (2021). Modified needleman–wunsch algorithm for clinical pathway clustering. *Journal of Biomedical Informatics*, 115:103668. DOI: 10.1016/j.jbi.2020.103668.

Breiman, L. (2001). Random forests. *Machine learning*, 45:5–32. DOI: 10.1023/a:1010933404324.

Che Mat, N. I., Jamil, N., Yusoff, Y., and Mat Kiah, M. L. (2024). A systematic literature review on advanced persistent threat behaviors and its detection strategy. *Journal of Cybersecurity*, 10(1):tyad023. DOI: 10.1093/cybsec/tyad023.

Chen, J., Hu, S., Zheng, H., Xing, C., and Zhang, G. (2023). Gail-pt: An intelligent penetration testing framework with generative adversarial imitation learning. *Computers Security*, 126:103055. DOI: 10.1016/j.cose.2022.103055.

Da Silva, F. L. and Costa, A. H. R. (2019). A survey on transfer learning for multiagent reinforcement learning systems. *Journal of Artificial Intelligence Research*, 64:645–703. DOI: 10.1613/jair.1.11396.

Disha, R. A. and Waheed, S. (2022). Performance analysis of machine learning models for intrusion detection system using gini impurity-based weighted random forest (giwrf) feature selection technique. *Cybersecurity*, 5(1):1. DOI: 10.1186/s42400-021-00103-8.

Farouk, M., Sakr, R. H., and Hikal, N. (2024). Identifying the most accurate machine learning classification technique to detect network threats. *Neural Computing and Applications*, 36(16):8977–8994. DOI: 10.1007/s00521-024-09562-9.

Gancheva, V. and Stoev, H. (2023). An algorithm for pairwise dna sequences alignment. In *International Work-Conference on Bioinformatics and Biomedical Engineering*, pages 48–61. Springer. DOI: 10.1007/978-3-031-34953-9$_4$.

Gangupantulu, R., Cody, T., Rahma, A., Redino, C., Clark, R., and Park, P. (2021). Crown jewels analysis using reinforcement learning with attack graphs. In *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–6. DOI: 10.1109/SSCI50451.2021.9659947.

Holm, H. (2022). Lore a red team emulation tool. *IEEE Transactions on Dependable and Secure Computing*, 1:1–1. DOI: 10.1109/TDSC.2022.3160792.

Horta, A., dos Santos, A. F. P., and Goldschmidt, R. R. (2024a). Evaluating the stealth of reinforcement learning-based cyber attacks against unknown scenarios using knowledge transfer techniques. *Journal of Computer Security*, (Preprint):1–19. DOI: 10.3233/jcs-230145.

Horta, A., Santos, A., and Goldshmidt, R. (2024b). Kill chain catalyst for autonomous red team operations in dynamic attack scenarios. In *Anais do XXIV Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*, pages 415–430, Porto Alegre, RS, Brasil. SBC. DOI: 10.5753/sbseg.2024.241371.

Ibrahim, M. K., Yusof, U. K., Eisa, T. A. E., and Nasser, M. (2024). Bioinspired algorithms for multiple sequence alignment: A systematic review and roadmap. *Applied Sciences*, 14(6):2433. DOI: 10.3390/app14062433.

Janisch, J., Pevnỳ, T., and Lisỳ, V. (2023). Nasimemu: Network attack simulator & emulator for training agents generalizing to novel scenarios. In *European Symposium on Research in Computer Security*, pages 589–608. Springer. DOI: 10.48550/arXiv.2305.17246.

Li, L., El Rami, J.-P. S., Taylor, A., Rao, J. H., and Kunz, T. (2022). Enabling a network ai gym for autonomous cyber agents. In *2022 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 172–177. IEEE. DOI: 10.1109/csci58124.2022.00034.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR. DOI: 10.48550/arxiv.1602.01783.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., *et al*. (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533. DOI: 10.1038/nature14236.

Ortiz-Garces, I., Gutierrez, R., Guerra, D., Sanchez-Viteri, S., and Villegas-Ch., W. (2023). Development of a platform for learning cybersecurity using capturing the flag competitions. *Electronics*, 12(7). DOI: 10.3390/electronics12071753.

Paudel, B. and Amariucai, G. (2023). Reinforcement learning approach to generate zero-dynamics attacks on control systems without state space models. In *European Symposium on Research in Computer Security*, pages 3–22. Springer. DOI: 10.1007/978-3-031-51482-1$_1$.

Poinsignon, T., Poulain, P., Gallopin, M., and Lelandais, G. (2023). Working with omics data: An interdisciplinary challenge at the crossroads of biology and computer science. In *Machine Learning for Brain Disorders*, pages 313–330. Springer. DOI: 10.1007/978-1-0716-3195-9$_1$0.

Pozdniakov, K., Alonso, E., Stankovic, V., Tam, K., and Jones, K. (2020). Smart security audit: Reinforcement learning with a deep neural network approximator. In *2020 International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA)*, pages 1–8. DOI: 10.1109/CyberSA49311.2020.9139683.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR. DOI: 10.48550/arxiv.1502.05477.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. Available at:`https://arxiv.org/abs/1707.06347`.

Sharma, P. and Rana, C. (2024). Artificial intelligence based object detection and traffic prediction by autonomous vehicles–a review. *Expert Systems with Applications*, page 124664. DOI: 10.1016/j.eswa.2024.124664.

Standen, M., Lucas, M., Bowman, D., Richer, T. J., Kim, J., and Marriott, D. (2021). Cyborg: A gym for the development of autonomous cyber agents. In *IJCAI-21 1st International Workshop on Adaptive Cyber Defense*. arXiv. DOI: 10.48550/ARXIV.2108.09118.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press, second edition. DOI: 10.1109/tnn.1998.712192.

Thangavel, K., Sabatini, R., Gardi, A., Ranasinghe, K., Hilton, S., Servidia, P., and Spiller, D. (2024). Artificial intelligence for trusted autonomous satellite operations. *Progress in Aerospace Sciences*, 144:100960. DOI: 10.1016/j.paerosci.2023.100960.

Tran, K., Akella, A., Standen, M., Kim, J., Bowman, D., Richer, T., and Lin, C.-T. (2021). Deep hierarchical reinforcement agents for automated penetration testing. In *IJCAI-21 1st International Workshop on Adaptive Cyber Defense*. arXiv. DOI: 10.48550/ARXIV.2109.06449.

Viering, T. and Loog, M. (2023). The shape of learning curves: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(6):7799–7819. DOI: 10.1109/TPAMI.2022.3220744.

Yang, Y. and Liu, X. (2022). Behaviour-diverse automatic penetration testing: A curiosity-driven multi-objective deep reinforcement learning approach. DOI: 10.48550/ARXIV.2202.10630.

Zhou, S., Liu, J., Hou, D., Zhong, X., and Zhang, Y. (2021). Autonomous penetration testing based on improved deep q-network. *Applied Sciences*, 11(19). DOI: 10.3390/app11198823.