Kill Chain Catalyst for Autonomous Red Team Operations in Dynamic Attack Scenarios

Antonio Horta[1,2], **Anderson dos Santos**[1,3], **Ronaldo Goldshmidt**[1]

[1]Military Institute of Engineering - IME
Praça General Tibúrcio, 80 - Urca, Rio de Janeiro - RJ, Brazil

[2]Accenture
Rio de Janeiro - RJ, Brazil

[3]Venturus Centro de Inovação Tecnológica
Campinas - SP, Brazil

antonio@horta.net.br,{anderson,ronaldo.rgold}@ime.eb.br

Abstract. *From the perspective of real-world cyber attacks, executing actions with minimal failures and steps is crucial 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 and low resilience in dynamic environments. Therefore, the Kill Chain Catalyst (KCC) has been introduced: an RL algorithm that employs decision tree logic, inspired by genetic alignment, prioritizing resilience in dynamic scenarios and limited experiences. Experiments reveal significant improvements in reducing steps and failures, as well as increased rewards when using KCC compared to other RL algorithms.*

1. Introduction

With the advancement of technology, tasks once performed by humans are now autonomously executed by machines embedded with artificial intelligence. 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) for enhancing the security of companies, training, competitions, and cyber warfare.

In the realm of cyber attacks, particularly in scenarios such as Capture the Flag (CTF) tournaments, attackers lack prior knowledge of the target environment, which unfolds during the campaign (Ortiz-Garces et al. 2023). According Che Mat et al. (Che Mat et al. 2024), a strategic emphasis on stealthy becomes crucial, underscoring a decisive sequencing decision-making process to link attack actions with fewer steps and fails, minimizing the likelihood of attack be exposed. Therefore, in light of the extensive damage wrought by cyber attacks in recent years, the exploration of autonomous cyber attacks holds paramount importance.

The literature reveals, Reinforcement Learning (RL), as an approach widely employed 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* involves a trial-and-error process wherein an agent gains knowledge from its environment through rewards or penalties, progressively improving its performance over time.

An attack involves crucial aspects to achieve their objectives, such as: resilience in dynamic environments, for example, a victim remediating a vulnerability during the attack and stealthy, in which, involves the minimization of steps and fails during the incursion. Despite, *RL* emerging as the predominant approach in autonomous cyber attack research. According Horta Neto et al. (Horta Neto et al. 2024), limitations arise in some attack scenarios, including high amount of fails during attack, the poor performance of *RL* agents with limited training data and the issue of low resilience in dynamic environments.

The existing studies predominantly focus on maximizing rewards to evaluate algorithms, overlooking the aspects of stealthiness and resilience. This work aims to rectify this deficiency by developing an *RL* algorithm that not only minimizes failures and steps, and works with limited data, but also is resilient in dynamic environments, with a primary focus on the sequencing of attacks from the perspective of real-world scenarios.

To attain this objective, the Kill Chain Catalyst (KCC) *RL* algorithm is introduced. *KCC* employs decision tree logic to guide the agent in attacks, enhancing resilience in dynamic environments and stealthiness through minimization of fails and steps. Additionally, a catalyst inspired by genetic alignment, optimizes the search for the most effective sequence chaining. The standout feature of *KCC* in sequential decision-making problems for cyber attacks lies in its use of *Random Forest* as the *RL* engine, in contrast to other algorithms that are neural network-based.

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 analyze the learning curve in terms of steps, rewards, and failures of *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 340.92% for steps, 138.29% for rewards, and 4585.11% for failures when performing attacks using *KCC* compared with algorithms such as *A2C*, *PPO*, *TRPO*, and *DQN*, which struggled to generalize attack sequences.

The article is structured to explore the topic comprehensively. It begins with an Introduction outlining the research objectives. The Background section covers essential concepts, used in this research. The Related Work section reviews relevant literature and contextualizes the study. The Methodology section details the *KCU* approach. The Experiments and Results and Discussion sections present the experimental setup, analysis, and findings. Finally, the Conclusion synthesizes the findings and suggests future research directions.

2. Background

The background section delves into various foundational concepts crucial to the study, including sequence alignment from genetic alignment, reinforcement learning, *Random Forest*, and *Gini Impurity-Based Weighted Random Forest*.

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 (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, composed of amino acids denoted by one-letter codes, play a crucial 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 Gacheva (Gancheva and Stoev 2023), is commonly employed. According Ibrahim et al. (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 path through the matrix. The scoring system incorporates match, mismatch, and gap penalties to effectively assess sequence similarity.

The 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 algorithm

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 (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.

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* plays a pivotal role. 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 (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* (GIWRF) 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, such as Equation 1.

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

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 2).

$$I_g(f) = \sum_M w_{p,n} \ \sum_\tau \Delta i_f(\tau, M) \tag{2}$$

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.

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. (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 main researches related to the theme discussed in this article counting with experiments involving automation of cyber attacks operations. It is organized in 6 columns, where: *Environment* detonates the type of environment used in the experiment, been among: *simulation* or *real*. *Algorithm* the reinforcement learning algorithm, *Analysis* denotes the variables analyzed in experiments and lastly, *Ref.* indicates the research in which the experiment was performed.

The analysis methodologies employed in the research shown in Table 2, primarily revolve around learning curves for steps and rewards. Failures in the kill chain during attacks were not considered, although some simulators, such as NASim[1], incorporate this variable in their simulations.

Considering all simulation environments shown in the Table 2, they are based on simulators and in hosts. None of the experiments in Table 2 employed techniques for reconnaissance, simulated dynamic environments, changing something in the state to analyze the implications, or analyzing fails during the attacks. These types of approaches are important for better understanding the behavior 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.

Table 2. Related works with experiments.

Environment	Algorithm	Analysis	Ref.
Real	A3C/DPPO/GAIL	Steps, Reward, Loss	(Chen et al. 2023)
Real	DQN	Steps, Rewards	(Li et al. 2022)
Real	Random Forest	Rewards	(Holm 2022)
Simulation	NDSPI-DQN dec.	Steps, Rewards	(Zhou et al. 2021)
Simulation	CLAP(PPO+RND)	Steps, Rewards	(Yang and Liu 2022)
Simulation	HA-DQN	Steps, Rewards	(Tran et al. 2021)
Simulation	DQN + LSTM	Steps, Rewards	(Standen et al. 2021)

4. Methodology

The section on Methodology introduces 3 significant components: The *Kill Chain Unscrambler*, *Kill Chain Catalyst* and the *Exoskeleton*[2], designed to address limitations in conventional *RL* algorithms when applied to sequencing tasks in stochastic environments, particularly within *CTF* scenarios.

4.1. Kill Chain Unscrambler

The Kill Chain Unscrambler *KCU* is specifically designed to overcome challenges in identifying attack sequences in attacks, where the target scenario is unknown, as opposed to those performed against simulated or emulated ones. In contrast with 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 networks refit 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 best patterns in term of steps size, undergoing a *KCU* transformation process

[1] `https://github.com/Jjschwartz/NetworkAttackSimulator`
[2] `https://gitlab.com/antonio50/exoskeleton`

from observed data to make predictions without relying on neural networks. The fit method in the *Tree classifier* class in which is using *Random Forest* for training the model.

The *KCU* class, extending the *BaseAlgorithm* from *Stable Baselines 3*[3], orchestrates the overall functionality of the *KCU* algorithm. It can be configured to use *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 catalytic 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 expedite the convergence of the learning curve within *KCU*, seeking sequences with a reduced number of steps, and crucially, minimizing fails in the chaining of attack techniques to achieve more stealthy and realistic attack sequences.

In this process, genetic alignment plays a pivotal 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, functioning akin to an enzymatic process to accelerate the search for optimal kill chain sequences.

4.2. KCU Agent

The Figure 1 illustrates the operational workflow of the *KCU* agent, specifically designed for executing Reinforcement Learning tasks in stochastic environments with sequencing requirements. This agent operates by ingesting input scenarios, which can originate from either simulators or real computing systems settings.

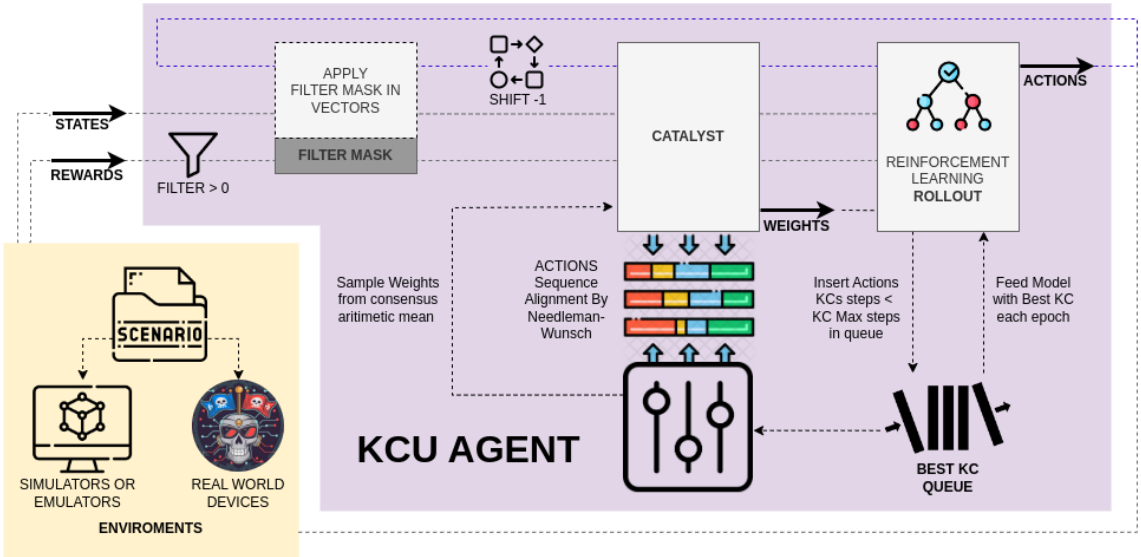


Figure 1. The Kill Chain Unscrambler with Catalyst process

[3]https://stable-baselines3.readthedocs.io

From these scenarios, the *KCU* agent receives (θ_t) denoting *States* ($\overrightarrow{S}_t$) and *Rewards* ($\overrightarrow{R}_t$) corresponding to the executed *Actions* ($\overrightarrow{A}_t$), represented as Equation 3. Considering the Equation 4, the agent then creates a mask ($\overrightarrow{J_t}$) based on the indices (j) of the reward vector ($\overrightarrow{R_t}$) that are greater than 0, indicating positive effects without penalties or failures. Beside *Reward* ($\overrightarrow{R}_t$), this mask is applied to both the *States* ($\overrightarrow{S_t}$) and *Actions* ($\overrightarrow{A_t}$) vectors according Equations 5, 6 and 7.

$$\theta_t = \overrightarrow{S}_t, \overrightarrow{R}_t, \overrightarrow{A}_t \quad (3) \qquad \overrightarrow{J_t} = j_{\overrightarrow{R_{t>0}}} \quad (4)$$

$$\overrightarrow{R_t} = \overrightarrow{R}_{tj}(j \in \overrightarrow{J}_t) \quad (5) \qquad \overrightarrow{S_t} = \overrightarrow{S}_{tj}(j \in \overrightarrow{J}_t) \quad (6) \qquad \overrightarrow{A_t} = \overrightarrow{A}_{tj}(j \in \overrightarrow{J}_t) \quad (7)$$

Then, according Equation 8, a shift of -1 is applied to the indexes of *Actions* ($\overrightarrow{A_t}$) vector.

$$\overrightarrow{A_t} = \overrightarrow{A}_{t(j\text{-}1)} \quad (8)$$

Next, the *Actions* ($\overrightarrow{A_t}$) vector undergoes a Catalysis process, where, in conjunction with the *Best KC QUEUE* ($\overrightarrow{QA}$), it undergoes sequence alignment using the *Needleman-Wunsch* (NW) (Gancheva and Stoev 2023) algorithm as shown in Equation 9.

$$KCC = mode(NW(\overrightarrow{A_t}, \overrightarrow{QA})) \quad (9)$$

Upon alignment, a consensus is reached through mode calculation, generating a consensus vector. The catalysis process concludes with the computation of the average consensus vector, which serves as the weight vector ($\overrightarrow{W_t}$) according Equation 10.

$$\overrightarrow{W_t} = \frac{\overrightarrow{KCC}_{tj}}{\sum \overrightarrow{KCC}_{tj}} \quad (10)$$

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 2). This policy formation uses the weight vector for each sample. If the resulting action sequence after rollout is shorter than the longest action vector in the *Best KC QUEUE* ($\overrightarrow{QA}$), it is inserted into the queue, which is then sorted in descending order based on the number of steps (Equation 11).

$$\overrightarrow{QA} = \overrightarrow{QA} \cup \overrightarrow{A}, if(len(\overrightarrow{A}_t) < maxlen(\overrightarrow{QA}) \quad (11)$$

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 hyperparameters 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 earlier 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 last step is equal the maximum steps per epoch and if last step exceeds the standard deviation from moving average of steps for the last 10 episodes, the buffer (QUEUE Best KC) 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

In the context of the objective, a server has been configured with the characteristics of a *CTF* challenge to establish the *Dummy* scenario for attack practice, using images from *vunlhub*[4]. This server operates within a container, running 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.

Regarding the integration with reinforcement learning, an interface named *Exoskeleton*[2] has been devised to facilitate autonomous attack processes. This interface operates at a low level of abstraction, enabling reinforcement learning algorithms to interact seamlessly with the target server. Within the *Exoskeleton*[2], the agent is equipped with eight *MITRE ATT&CK*[5] 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*[5] techniques implemented in the *Exoskeleton*[2] 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. The complete references and detailed descriptions of these techniques can be accessed on the *MITRE ATT&CK*[5].

[4]`https://github.com/vulhub/vulhub/tree/master/libssh/CVE-2018-10933`
[5]`https://attack.mitre.org/matrices/enterprise/`

Table 3. Exoskeleton's action space

Tactic id	Tactic name	Technique id	Technique name	Cmd or 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

Building upon the outlined techniques, the shortest kill chain available for exploiting the *Dummy* scenario in the *Exoskeleton*[2] is depicted in Figure 2. Kill chain A (KC A) represents a scenario with security patches updated, characterized by a minimal number of steps and failures. It includes reconnaissance activities like 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 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, the data collection from the local system culminates in achieving the objective of capturing the flag. In contrast, Kill chain B (KC B), illustrated in Figure 2, 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*[6].

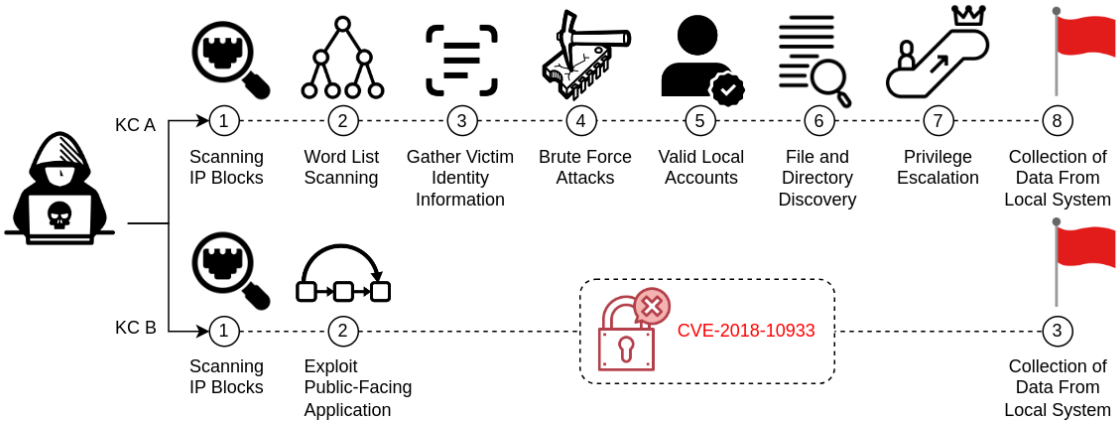


Figure 2. Kill chains available in Exoskeleton's Dummy Scenario.

Exoskeleton[2] transforms real computing systems into a *Gymnasium*-compatible[7] scenario format, a platform tailored for testing reinforcement learning algorithms.

Integral to the reinforcement learning process, the *Exoskeleton*[2] adopts a reward model. Successful actions resulting in state modifications are rewarded with

[6]`https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-10933`
[7]`https://github.com/Farama-Foundation/Gymnasium`

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. This intricate reward system provides the reinforcement learning agent with a clear incentive structure as it navigates the challenges presented by the *CTF* environment.

4.3.2. State Space

The *Exoskeleton*^{2} 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 4. Table 4 organizes the fields into two columns: the first delineates the field, and the second furnishes a succinct description.

Table 4. 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.

The invariant model proposed by Janisch et al. (Janisch et al. 2023), which processes each host feature vector individually with a shared embedding function, served as the basis for the modeling of *Exoskeleton*^{2}. According Janisch et al. (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*^{2} transforms the *State* into a vector of integers representing *Observables*. The modeling of *States* and *Observables* allows infinite scalability for *States*. This modeling become *Actions* techniques independent of the variables in the scenario. The independence generated by this modeling is an important factor in distinguishing *Exoskeleton*^{2} from simulators employed in related work.

5. Experiments

The experiment involving *KCC* will be assessed alongside four algorithms: the traditional value-based *Deep Q-Network* (DQN), policy-based *Proximal Policy Optimization* (PPO), *Trust Region Policy Optimization* (TRPO), and the hybrid *Advantage Actor-Critic* (A2C) *RL* algorithms. While these algorithms use neural networks as their policy, *KCC* employs decision tree logic. The assessment will focus on key metrics such as the learning curve, total rewards, total steps, and total failures.

To analyze the performance of *KCC* in attack operations, considering the learning curve for limited attack experiences and ability to handle dynamic scenar-

ios, 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: *PPO, TRPO, A2C*, and *DQN*. 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 rectified midway through each algorithm's learning cycle, prompting the converged algorithms to explore new attack paths.

The experiment involves conducting attacks using reinforcement learning agents with different algorithms in a *CTF* competition. In this competition, 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 trial, the *Dummy* scenario introduces an additional vulnerability, *CVE-2018-10933*[6], 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 5 were considered. Parameters not explicitly provided will follow the default values of the *Stable Baselines 3 framework*.

Table 5. Essays' parameters

Agent	Parameter	Essay 1	Essay 2
KCC, PPO, TRPO, A2C, DQN	epochs	50	50
KCC, PPO, TRPO, A2C, DQN	max steps per epoch	50	50
KCC, PPO, TRPO, A2C, DQN	timesteps	2500	2500
KCC	epsilon (e-greedy)	0.25	0.25
KCC	decay_rate (e-greedy decay rate)	0.01	0.05
KCC	kcc (catalyst)	True	True
KCC	seed (random seed)	1	1
KCC	buffer (size of the best KCs QUEUE)	20	20
KCC	n_estimators	50	50
KCC	min_samples_leaf	1	1
KCC	dynamic_scenario	False	True

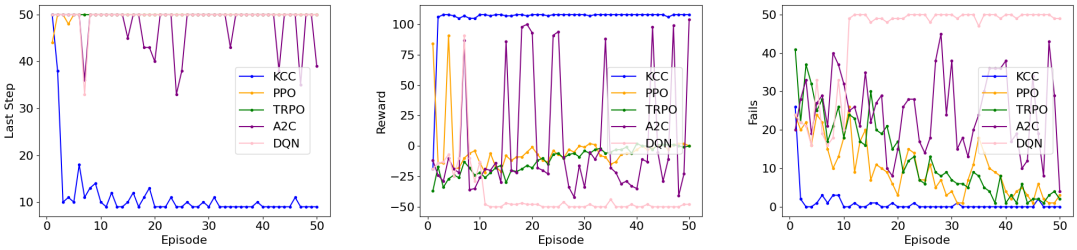
6. Results and Discussion

For the first essay, *KCC* was compared with traditional *RL* algorithms in the *CTF* scenario. As previously mentioned, in real attack situations, all algorithms were subjected to a scenario with limited steps and epochs for learning. Table 6 presents the cumulative values and percentage differences of each algorithm, comparing them with the results achieved by *KCC*. *KCC*, utilizing decision tree logic in contrast to the neural networks used by the other algorithms, demonstrated superior overall performance in this type of problem. *KCC* accumulated 567 steps by the end of the learning cycle, compared to the others, which remained around 2500 steps. However, when analyzing cumulative failures and rewards, distinctions among the algorithms become evident. *KCC* stands out as the best, followed by *A2C*, *PPO*, *TRPO*, and *DQN* for rewards, and *PPO*, *TRPO*, *A2C*, and *DQN* for failures.

Table 6. Cumulative Values and Percentage Differences Relative to KCC

Serie	Steps	Diff Steps	Rewards	Diff Rewards	Fails	Diff Fails
KCC	**567**	**0.00%**	**5252**	**0.00%**	**47**	**0.00%**
PPO	2492	339.51%	-162.0	103.08%	496	955.32%
TRPO	2500	340.92%	-567.0	110.80%	640	1261.70%
A2C	2383	320.28%	184.0	96.50%	1241	2540.43%
DQN	2483	337.92%	-2011.0	138.29%	2202	4585.11%

As shown in Figures 3.a, 3.b and 3.c only *KCC* generalized, achieving an 8-step kill chain. This serves as evidence that these algorithms are learning and require more experiences to converge. This essay makes clear the challenges faced by *RL* algorithms based on neural networks when dealing with limited data for training, a factor present in sequencing of real attack situations.

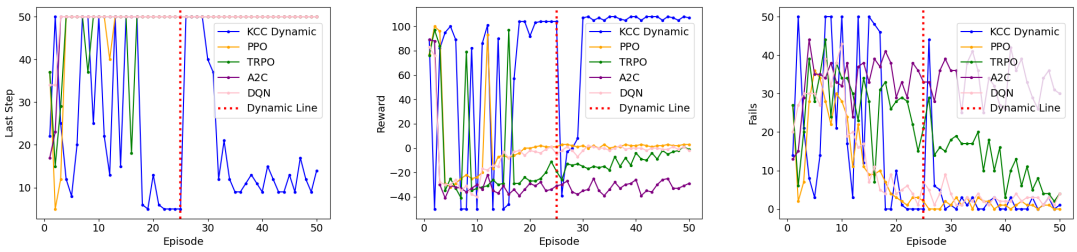


(a) Last Step x Episode (b) Reward x Episode (c) Fails x Episode

Figure 3. Limited experiences for KCC, A2C, PPO, TRPO and DQN

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 2. However, during the ongoing learning cycle at step 25, a remediation ensued, restricting access solely to *KC A*, also depicted in Figure 2. This deliberate protocol aimed at scrutinizing agent behavior concerning resilience in dynamic scenarios.

Figure 4 encapsulates the comprehensive evaluation of agents subjected to dynamic scenarios. The red vertical demarcation (dynamic line) signifies the vulnerability-fixing juncture . The *KCC* agent, configured for dynamic scenarios, manifests discernible alterations in strategy upon vulnerability resolution. There-



(a) Last Step x Episode (b) Reward x Episode (c) Fails x Episode

Figure 4. Comparing KCC, A2C, PPO, TRPO and DQN in a dynamic scenario

fore, *KCC* dynamically adapts its approach, recalculating strategies post-resolution, and resumes convergence after a dozen steps.

In contrast, other algorithms exhibit inconspicuous behavioral shifts post-vulnerability correction. This arises due to their proclivity for challenges within low-experience scenarios; not having converged to a kill chain at the time of the fix, they persist in seeking convergence pathways. Conversely, *KCC*, with the dynamic scenario's flag active, monitors learning evolution. Upon detecting any anomalies, it promptly recalibrates its exploitation strategy. Consequently, resilience factors remain indiscernible for other algorithms, as their quest for convergence is uninterrupted by the vulnerability fix.

Finally, the *KCU*, supported by the *KCC*, represents a significant step forward in autonomous cyber attack strategies. Its ability to outperform traditional *RL* algorithms in dynamic scenarios underscores the potential of genetic alignment techniques in cyber security. Future research should aim to refine these methods, explore broader applications, and address practical deployment challenges to fully realize the benefits of this approach.

7. Conclusions

This conclusion synthesizes the main findings and contributions of the research, highlighting the development and evaluation of the *KCC* algorithm, driven by an *RL* agent in attacks on dynamic scenarios. The *KCC*, utilizing decision tree logic and a catalyst inspired by genetic alignment, proved effective in overcoming limitations found in traditional neural network-based algorithms such as *PPO*, *TRPO*, *A2C*, and *DQN*, particularly in environments with limited learning experiences.

Experiments conducted within the context of a *CTF* tournament demonstrated the superiority of *KCC* in generalizing and optimizing attack sequences, minimizing the steps and failures required to achieve objectives. The results showed differences of up to 340.92% for steps, 138.29% for rewards, and 4585.11% for failures when performing attacks using *KCC* compared with other traditional *RL* algorithms. Moreover, *KCC*'s ability to quickly adapt to environmental changes, such as the remediation of vulnerabilities, highlighted its resilience and efficacy in dynamic scenarios, a feature not observed in other tested algorithms.

These results underscore the significance of decision tree-based approaches and the potential of the catalyst to enhance the performance of *RL* in cyber attacks. The research indicates that future studies should focus on refining the exploration phase of the algorithm, particularly given its stochastic nature, as this is crucial for the type of study conducted. Such refinement could not only optimize the achievement of performance benchmarks but also improve the offset when compared to other algorithms. Moreover, the integration of additional artificial intelligence techniques could further enhance the effectiveness and resilience of autonomous cyber defense systems. Conducting experiments in increasingly complex and variable environments could also provide further insights into the robustness and adaptability of the *KCU* in dynamic attack scenarios.

References

[Al-Azzawi et al. 2024] 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.

[Breiman 2001] Breiman, L. (2001). Random forests. *Machine learning*, 45:5–32.

[Che Mat et al. 2024] 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.

[Chen et al. 2023] 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.

[Disha and Waheed 2022] 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.

[Farouk et al. 2024] 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.

[Gancheva and Stoev 2023] 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.

[Gangupantulu et al. 2021] 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.

[Holm 2022] Holm, H. (2022). Lore a red team emulation tool. *IEEE Transactions on Dependable and Secure Computing*, 1:1–1.

[Horta Neto et al. 2024] Horta Neto, A. J., dos Santos, A. F. P., and Goldschmidt, R. R. (2024). Evaluating the stealth of reinforcement learning-based cyber attacks against unknown scenarios using knowledge transfer techniques. *Journal of Computer Security*, (Preprint):1–19.

[Ibrahim et al. 2024] 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.

[Janisch et al. 2023] 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.

[Li et al. 2022] 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.

[Mnih et al. 2016] 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.

[Mnih et al. 2015] 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.

[Ortiz-Garces et al. 2023] 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).

[Paudel and Amariucai 2023] 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.

[Poinsignon et al. 2023] 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.

[Pozdniakov et al. 2020] 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.

[Schulman et al. 2015] 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.

[Schulman et al. 2017] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms.

[Standen et al. 2021] 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.

[Sutton and Barto 2018] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press, second edition.

[Tran et al. 2021] 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.

[Yang and Liu 2022] Yang, Y. and Liu, X. (2022). Behaviour-diverse automatic penetration testing: A curiosity-driven multi-objective deep reinforcement learning approach.

[Zhou et al. 2021] 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).