Red Team LLM: towards an adaptive and robust automation solution

Christophe Genevey-Metat, Dorian Bachelot, Tudy Gourmelen, Adrien Quemat, Pierre-Marie Satre, Loïc Scotto, Di Perrotolo, Maximilien Chaux, Pierre Delesques, Olivier Gesny

Red Team LLM: towards an adaptive and robust automation solution

Christophe Genevey-metat
R&D AI / Cyber Team
Silicom
Rennes, France
cgeneveymetat@silicom.fr

Dorian Bachelot
R&D AI / Cyber Team
Silicom
Rennes, France
dbachelot@silicom.fr

Tudy Gourmelen
R&D AI / Cyber Team
Silicom
Rennes, France
tgourmelen@silicom.fr

Adrien Quemat
R&D AI / Cyber Team
Silicom
Rennes, France
aquemat@silicom.fr

Pierre-Marie Satre
R&D AI / Cyber Team
Silicom
Rennes, France
pmsatre@silicom.fr

Loïc Scotto Di Perrotolo
R&D AI / Cyber Team
Silicom
Rennes, France
lscottodiperrotolo@silicom.fr

Maximilien Chaux
R&D AI / Cyber Team
Silicom
Rennes, France
mchaux@silicom.fr

Pierre Delesques
R&D AI / Cyber Team
Pr0ph3cy
Guyancourt, France
pdelesques@pr0ph3cy.com

Olivier Gesny
R&D AI / Cyber Team
Pr0ph3cy
Rennes, France
ogesny@silicom.fr

***Abstract*—Artificial intelligence has become really popular in recent years, especially its embedding in cybersecurity applications. Today, studies have shown that reinforcement learning agents are able to find the optimal sequence of actions in order to attack a network. However, these agents are often over-trained and can neither adapt nor be robust to different networks from the ones they were trained on. We propose a new agent based on a zero-shot approach that adapts itself to any given network and that is robust to parameters and objectives changes without requiring another training phase. We introduce a new metric that better measures the ability of agents to attack a network without prior knowledge. In this paper, we also discuss about the first steps towards explainability for our model and its future improvements.**

***Index Terms*—pentesting automation, attack simulation, zero-shot classification, transformers, large language model, adaptability, robustness, explainability**

I. Introduction

Pentesting is an operation that consists in discovering the weaknesses of a network that an attacker could use to harm a company. In the research field, pentesting is often seen as an optimisation problem where the goal is to compromise a machine inside a network within a minimum number of actions. The actions are unit attacks that can be chosen by the agent. Several researchers [1], [3], [11], [14], [16] have already studied the use of reinforcement algorithms to solve this optimisation problem and several simulators, such as NASim [6], or Cyborg [8], have been developed to facilitate working on such algorithms. In a real-world situation, when an intruder attacks a network, having access to the entire network's arrangement beforehand is quite rare. Indeed, each network has its own topology with its own layout of subnetworks (subnets) and machines. As a result, an attacker needs to navigate between exploring the network in search of information about the target, exploiting machines and in some cases, bypassing firewall's restrictions. Thus, we believe that pentesting cannot be considered like a simple optimisation problem due to the presence of changing sets of actions and heavy interdependence between obtained answers and new actions at each step. The simulator NASim [6] developed by J. Schwartz is one of the most complex simulators that exist and represents well certain aspects and constraints that a real attacker may encounter. We developed a variant of NASim that can produce a natural textual representation of the network used by our model. We also use this modified NASim to calculate our own metric that better represents the agent's ability to use discoStovered information before exploiting an unknown machine.

In this paper, we use a pre-trained zero-shot classification model [15]. We have developed a specific textual observable environment to provide the model with a better level of adaptability compared to other reinforcement learning algorithms. Thus, this model is adaptive and able to attack other networks, unseen during the training phase. This model uses textual observables to take the most promising action. In these textual observations, we provide the goal of the challenge, some rules that the model has to follow to complete the challenge and information about the state of the targeted information system (e.g. machine state). We have also created a naive metric for the training and evaluation phase in order to know if the agent has taken the correct action thanks to its past knowledge, meaning that it was based upon useful information gathered prior to the execution. Or if it did by chance, implying that the attack is successful but without the agent exploiting the gathered information. For example, this naive metric allows the agent to be rewarded if it discovers that a certain vulnerable service is running on a machine and attacks this machine. We compare the performances of our reinforcement learning (RL) zero-shot model with other reinforcement learning algorithms (DQN, PPO recurrent, and CLAP) and discuss the results with our own metric further in the paper. In addition to this comparison, we also provide a preliminary explainability of our model based on the Shap

value [5]. We demonstrate our agent's confidence based on the observable textual environment (feasible actions included), as well as the importance of certain words in the context that allow the agent to take the right action. Since our model uses textual information, it is possible to extract a certain degree of explainability. Yet still limited, it opens up the discussion for more.

II. Related works

Schwartz and Kurniawati [6] proposed a benchmark environment to train and evaluate RL agents in the context of pentesting. Their benchmark environment called "NASim" allows them to simulate a network with various subnets and several machines per subnet. They include services, processes and operating systems in the machine information. Their environment also includes some restrictions of services to support several types of firewall configurations. They propose a list of actions that an RL agent can perform. Each action has a certain cost (due to its level of furtiveness) and a certain probability of success (for a non-deterministic aspect). This environment is a pure simulation but it includes some constraints found in the real world, these are discussed in section III-A.

Another benchmark environment exists called CybORG [9] developed for the CAGE (Cyber Autonomy Gym for Experimentation) challenges from the Department of Defense of the Australian government. This environment is better equipped than NASim and able to support simulations as well as emulation. It is even cited in the gap considerations page of NASim as an alternative for more thorough tests. However, we chose to go with NASim for several practical reasons. Firstly, the emulation part of CybORG was of no interest for us at this point and is not open sourced. Secondly, it was easier to compare our model to existing techniques. Indeed, most of the RL algorithms published up to this day use NASim and its predefined topologies and scenarios for benchmarking.

In addition to the proposed NASim environment, Schwartz [6] shows the performances of classical RL agents (DQN, Tabular Q-learning) on miscellaneous simple scenarios. They establish the performance of these RL agents against the number of machines in the network. They demonstrate how the performance decreases as network complexity increases, and that after a certain number of machines, a classical RL agent obtains a negative mean episodic reward.

Zhou and Liu [16] propose an improvement of the deep Q-network named NDSPI-DQN to solve the issue of sparse rewards and large action space problems. Their model uses soft Q-learning, duelling architecture, prioritised experience replay, and intrinsic curiosity to improve the exploration efficiency. They show that their model converges better than the baseline DQN, and can achieve a positive average episodic return, while the original DQN achieves a negative average episodic return.

Yizhou and Xin [14] propose a variant of the PPO algorithm called CLAP that can handle multi-objective reinforcement learning in a pentest context. They study the multi-objective in order to provide a diversity of attacks and paths that can compromise the network's security. CLAP uses a coverage mask mechanism that allows the model to keep track of previous actions taken in the past. This mechanism encourages the model to trigger an action based on both the previous action and the current observation. The Clap model is trained on three NASim scenarios described in table I. These three scenarios have different topologies and objectives. They show that their model converges quickly to the optimal sequence on the three scenarios during the training phase compared to other algorithms (DQN, Improved-DQN, HA-DQN). However, the authors only show the performance during the training phase and not for an evaluation phase. Thus, it raises questions about the adaptability of their model regarding other networks and during an evaluation phase.

TABLE I
NETWORK SCENARIOS USED BY CLAP MODEL.

	Subnets	Hosts	OS	Services	Processes
Small-linear	7	8	2	3	2
Medium	6	16	2	5	3
Large	8	23	3	7	3

In section III, we present the challenge for a RL agent to attack a network without prior knowledge, the process of decision-making, the observable environment developed for our model, and our own metric. In section IV, we present the performance of our model compared to other RL agents (DQN, PPO recurrent, and CLAP), and a preliminary study to observe the impact of certain words in the decision-making process.

III. Preliminary

In this section, we present the challenge of pentesting, the process of decision-making of our agent inside an adapted NASim environment, and our new metric for the training and evaluation.

A. Challenge of pentesting

From our point of view, the role of a Red Team is complex and is not limited to compromising a specific machine in a known information system as quickly as possible.Indeed, a RL agent does not have any prior knowledge about the target network. It operates on a partially observable environment, and its action space expands because a list of new actions appears when it discovers a machine.

We identify other constraints that an RL agent must take into account due to no prior knowledge about the target network. These following constraints are not specific to a RL agent and also exist for other machine learning agents that perform pentesting:

- Machine choice: the agent must be able to select the right machine to attack. If it discovers the target machine then it has to focus onto it, otherwise it should focus on the other machines of the network. For example, the machine

identifier may change from one network to another, as may its operating system or services.
- Attack choice: the agent needs to be able to select the right attack. If it doesn't have enough information to do so, it should keep going with the discovery stage; but if it does have enough, it should exploit the machine.
- Probability of success: an action has a certain probability of success. If the agent selects an action that fails, it needs to try again or select another action. The probability of success can vary from one network to another.
- Restriction of services: an action can fail due to some restrictions from the system. The agent needs to be able to select another path in order to compromise the machine, should a restriction happen. In addition, the agent must be able to make errors in order to discover this restriction. The restriction of services can also vary from one network to another.
- Choice of parameters: the agent must have the ability to select the right action based on the information discovered on the machine and the privilege levels available after the attack. The privilege levels on one machine can vary from one network to another.

These constraints may seem simple to manage for a human operator, but represent a challenge for RL agents because most of the time, RL agents will just repeat the optimal sequence they learned during training. If the network changes and these constraints also change, the RL agent will not be able to act correctly. This is why we believe that pentesting cannot be considered as a simple optimisation problem. We decided to choose the NASim environment because it includes all of these constraints. It can therefore be seen as a first pentesting environment for RL agents to work on.

B. Decision-making

Our approach is based upon the work of *Yin et al.* [15]. They propose a method for using a pre-trained sequence-to-sequence model and adapt it to a zero-shot classifier. Their model takes as input a sequence to be classified and generates a hypothesis from each candidate label. Their method can be used with different large pre-trained models such as BART [4] or Roberta [2] which are Transformers networks. We chose to focus on the BART model and slightly adapt their approach to play with a reinforcement learning environment. The Figure 1 illustrates the path and reasoning of our model when playing in reinforcement learning environment, and especially in the NASim environment [6]. Our model, in order to make a decision, works with two main types of sub-decision: it first chooses the target machine to attack, then it selects the action to perform on it. The "Overall context" is a description of the goal to achieve supported by some advice. The "machine available" is a description of the status of each machine. It represents the candidate labels. Thus, our model selects one of these candidates based on the overall context. After selecting the machine, our model must choose the action to perform on it. The "Machine's context" is a description of the current goal and the information found on the machine (during the previous iteration). The "Available actions" is a description of the available attacks that can be launched by our model. It is the new candidate labels that our model will select based on the context of the machine. After selecting the action, it is performed on the NASim environment and new "Overall context", "machine available", "Machine's context" and "Available actions" are generated.

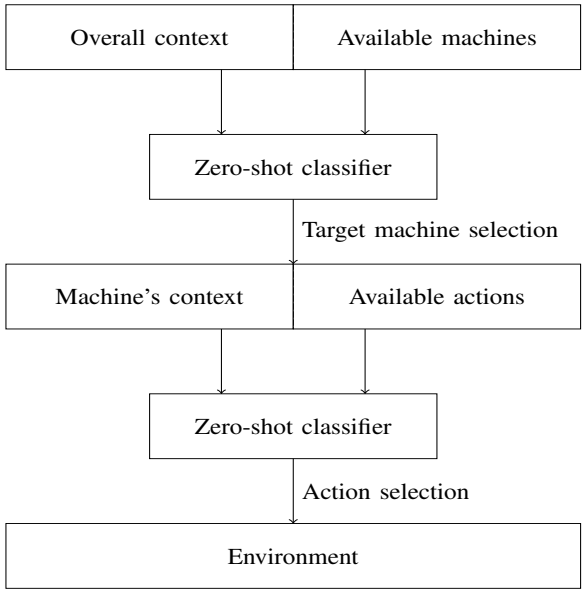

Fig. 1. Decision path taken by our model

C. Textual observables for decision support

In order to help our zero-shot model to select the right target machine and the correct action, we create a certain global context, a machine context, an action description and some textual observables for decision support.

1) Overall context: In the overall context, we store information about the potential path to the goal, a description of the goal to reach, and rules that the zero-shot must follow. The potential path to the goal is regularly updated by removing discovered and exploited machines from the list of potential paths. This textual observable helps the agent to focus solely on machines not yet exploited. The objective is also updated if it is composed of several sub-goals and if one of them is achieved. Therefore, the agent is able to focus on the second objective when the first one is accomplished. Unlike the other textual observables, the rules that the zero-shot must follow do not change. This element helps the agent to find another path when a restriction appears, it also gives greater confidence in the agent's decision-making. In section IV-B, we will talk more about this confidence in the decision making process.

Example of 'Overall context': *B,C,D,E,F are potentially linked to F. The goal is to get privileges on F while remaining undetected. We encourage to keep attacking machines where you do not have full access.*

2) Available machines: In the "Available machines" part, there is a descriptive list of all the discovered machines' statuses. The status of the machine is updated on certain conditions. Table II describes the conditions that make the candidate labels changes. The conditions and labels we have developed give the agent a better representation of the environment helping it in making the right decision at all times. We have chosen these labels in order to obtain decisions as deterministic as possible. The impact of the words used in these labels will be developed in section IV-B.

Example of 'Available machines': *A has already been attacked, B is a new machine, C is a new machine, D is a new machine, E is a new machine, F is one of the target machine.*

Conditions	Candidate labels (Actions)
Agent does not execute any action on A	A is a new machine
Agent executes at least one action on A	A has already been attacked
Agent discovers that A is the target machine	A is one of the target machine
Agent obtains root privileges on A	A is totally corrupted
Agent executes at least one action and receives a permission error	A is a dead end

TABLE II
CONDITIONS TO UPDATE THE CANDIDATE LABELS THAT CAN BE PRESENT IN THE LIST OF 'AVAILABLE MACHINE'

3) Machine context: Inside the machine context we have information about the goal, the services, the processes, and the operating system present on the machine. It also states if the machine is compromised or not. The goal information is updated as in the overall context. Most of this information is also used by the other RL agents (DQN, CLAP) from the state-of-the-art papers, except for the goal information part. Table III describes the observables that represent all information about a machine, and how it is updated after the agent triggers an action. We only illustrate the update via a specific sequence to give an overview, and to highlight the impacted data.

Example of 'Machine context': *The goal is to get privileges on F while remaining undetected. No service found on F. No operating system found on F. No process found on F. No privilege obtained on F.*

4) Available actions: In the available actions, we have information about actions that can be executed on the targeted machine. Table IV describes each action present in our scenarios of NASim: Tiny, Small-linear, and Medium (defined in the Table V). Currently, we have created 12 textual labels that allow playing on different scenarios of NASim but not all. Some of these actions also target an operating system (Linux, or Windows) in addition to the machine, and we take this into account in our labels. We have also added information to some actions for the agent to execute them when a certain status of a machine appears. This is particularly true for the following actions: "process_scan", "subnet_scan", "pe_daclsvc", "pe_tomcat", and "pe_schtask" which will be executed once the machine is infected.

Conditions	Observables
The agent does not execute any action on A	No service found on A. No operating system found on A. No process found on A. No privilege obtained on A.
The agent executes 'service_scan' on A	**The service ssh is running on A**. No operating system found on A. No process found on A. No privilege obtained on A.
The agent executes 'os_scan' on A (in addition to the previous)	The service ssh is running on A. **The operating system linux is running on A**. No process found on A. No privilege obtained on A.
The agent executes 'e_ssh' on A (in addition to the previous)	The service ssh is running on A. The operating system linux is running on A. No process found on A. **A is finally infected**.
The agent executes 'process_scan' on A (in addition to the previous)	The service ssh is running on A. The operating system linux is running on A. **The process tomcat is running on A**. A is finally infected.

TABLE III
CONDITIONS TO UPDATE THE OBSERVABLES THAT CAN BE PRESENT IN A 'MACHINE CONTEXT'

NASim Actions	Candidate labels (Actions)
service_scan	reveal service on A
os_scan	reveal operating system on A
process_scan	reveal process by infecting A
subnet_scan	discover subnet by infecting A
e_ftp [linux,windows]	exploit ftp [linux,windows] on A
e_http [linux,windows]	exploit http [linux,windows] on A
e_ssh [linux,windows]	exploit ssh [linux,windows] on A
e_samba [linux,windows]	exploit samba [linux,windows] on A
e_smtp [linux,windows]	exploit smtp [linux,windows] on A
pe_daclsvc	get highest privileges by infecting A with daclsvc windows
pe_tomcat	get highest privileges by infecting A with tomcat linux
pe_schtask	get highest privileges by infecting A with schtask windows

TABLE IV
LABELS USED TO REPRESENT THE NASIM ACTIONS

Example of 'Available actions': *reveal service on F, discover subnet by infecting F, reveal process by infecting F, exploit ssh linux on F, exploit http on F, exploit ftp windows on F, get highest privileges by infecting F with tomcat linux, get highest privileges by infecting F with daclsvc windows.*

D. Evaluation metric

As presented in section III-A, we suggest that measuring the performance of a RL agent based on the number of actions done to compromise the target machine is not an efficient metric because it does not encourage actions that reflect the behaviour of a human pentester confronted with an IS that he does not know. Indeed, pentesters tend to want to be as discreet as possible and to obtain as much useful information about a machine as possible before attacking it. We choose to evaluate the ability to attack a network with a new metric that encourages the RL agent to discover the necessary information before attacking a machine.

At the end of the challenge, we return the basic reward plus a bonus. Equation 1 describes this new metric which allows to better evaluate the performance of our model during the training and evaluation phase. The basic reward called R_{base} is the sum of the cost of all the actions carried out. The bonus is the sum of A_c^t (multiplied by a factor α) which is the one-hot incorporation of the action taken by the agent at time t after having discovered the right information. α is a configurable factor that compensates for the cost of discovery actions. RL agent using information from discovery actions to exploit machines obtain reward bonuses. This factor must be higher than 3 because the attacker needs to discover at least three elements per machine: operating system, services, processes and each discovery cost -1. This metric is used with all agents (CLAP, DQN, PPO, Zero-shot) and we have a safeguard that prevents discovery actions that have already been positively rewarded from being rewarded again.

$$R_{final} = R_{base} + \sum_{i=1}^{t} A_c^t * \alpha \tag{1}$$

IV. Contributions

A. Experiments on NASim

In this section, we present the performances of our zero-shot model compared to the PPO recurrent network and the CLAP model.

TABLE V
NETWORK SCENARIOS USED BY OUR MODEL.

	Subnets	Hosts	OS	Services	Processes
Tiny	4	3	1	1	1
Small-linear	7	8	2	3	2
Medium	6	16	2	5	3

We also present the performances of our model with our new metric used during the evaluation phase. Refer to Appendix 10 for a comparison of the performance of our model against other Reinforcement Learning agents, using the traditional reward.

For the experiments, we choose to compare the DQN, PPO recurrent, and CLAP with our model. The DQN and CLAP were chosen because they are mentioned in several state-of-the-art papers [6], [14]. We also decided to add the PPO recurrent since we wanted a classical RL agent that can handle partially observable environment. We had to train the other models in order to compare them with ours. This training took 100,000 steps for the DQN and PPO recurrent models, and 5,000,000 for the CLAP model. It is important to note that our model did not require any training at all because it is a pre-trained model, therefore we haven't re-trained it on the NASim environment. We have noticed that the DQN and PPO recurrent do not converge after 100,000, this is why the training steps are set to this value. For CLAP, we took the same hyperparameters as the authors chose [14].

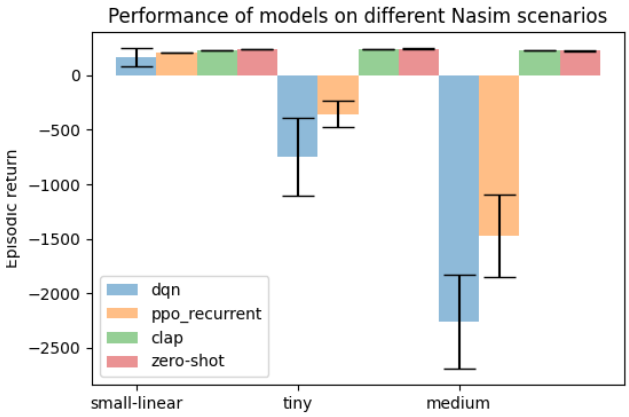

Fig. 2. Comparison of performances from RL agents (DQN, CLAP, PPO recurrent and zero-shot) trained and evaluated with the new evaluation metric

Figure 2 shows the performance of our zero-shot model compared to other RL agents (CLAP, PPO recurrent). The episodic return is the final reward given to the agent at the end of the challenge. This reward is computed with the new metric evaluation introduced in the previous section. We compare all algorithms on three scenarios (tiny, small-linear, and medium) using a mean episodic return. It is the mean value obtained after five runs.

In the following results 2, we can see that the DQN, PPO recurrent and CLAP obtain good results. However, we would like to remind that these models are not adaptive and they are evaluated on the same network they trained on. Whereas our zero-shot approach does not require training and therefore does not know the targeted network in advance.

For the tiny scenario (on the left-hand side of the figure 2), we observe that the four models have positive episodic returns. The DQN model has a mean episodic return of 163, the PPO model gets a mean episodic return equal to 207, the CLAP model obtains a mean episodic return of 230, and the zero-shot model reaches a mean episodic return equal to 239. Thus, our model outperforms the DQN, PPO recurrent and CLAP network, even if these models have seen the network during the training phase.

For the small-linear scenario (on the middle of the figure 2), we observe that only the CLAP and zero-shot models have positive episodic returns. The DQN model reaches a mean

episodic return of –746, the PPO model has a mean episodic return equal to -355, the CLAP model obtains a mean episodic return of 235, and the zero-shot model gets a mean episodic return equal to 244. The DQN has difficulties to converge due to the partially observable environment and the complexity of the network. The PPO recurrent obtains a better score than the DQN thanks to its recurrent layer that better handles partially observable environment. Finally, we observe that our model is 61 points better compared to the CLAP model.

For the medium scenario (on the right-hand side of the figure 2), we also observe that only CLAP and zero-shot models have positive episodic returns. The DQN model reaches a mean episodic return of -2260, the PPO model obtains a mean episodic return equal to -1471, the CLAP model has a mean episodic return of 234, and the zero-shot model gets a mean episodic return equal to 225. We observe that CLAP outperforms our model, our model is -9 points worst compared to the CLAP model. In this case, CLAP performs better, which is not surprising, because it should be pointed out that it saw the network during its training phase, whereas our model did not.

These results show that our zero-shot approach outperforms other RL agents (PPO and CLAP) in terms of mean episodic returns. We can conclude that our model can handle different networks without needing a new training.

B. Towards explainability

We are aware that the work we present cannot lead to a complete explainability of our model. However, this work has allowed us to better understand the importance of the words that are given to the algorithm. The choice of words also helped in the decision making process to get more accurate and deterministic answers. This is why we present, in this section, a preliminary study on the importance of words in decision making.

We start by looking at the confidence that our model gives to each candidate labels. We then look at the Shap value whose interest is to explain the influence of inputs on model outputs. In our case, we can thus see the influence of the input words on the decision-making of the actions (selection of the machine and selection of the action).

1) Machine selection: In this section, we will show how the words in the list of available machines impact the decision of our models. In order to illustrate this, we have created five distinct contexts called ”Context 1” to ”Context 5” and three candidate labels per context. These new contexts and candidate labels are illustrated and described in the Table VI and IX. In the Figure 3, we can observe the probability of each action A1, B1, C1 for the different contexts. In the context 1, we can select three new machines represented by the three actions A1, B1, C1. There is no advantage in selecting one over the other, this is why we can observe close probabilities between actions A1, B1, and C1. Thus, our model decides to choose the machine A (by selecting the action A1). In the context 2, the agent has previously launched one action on the machine A. Therefore, we can observe that the action A1

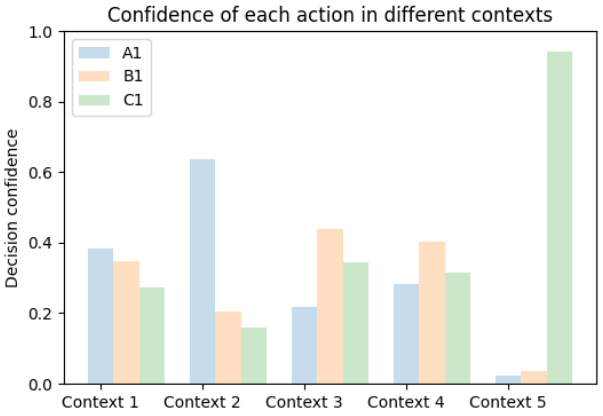

Fig. 3. Decision confidence for different contexts about selecting the machine

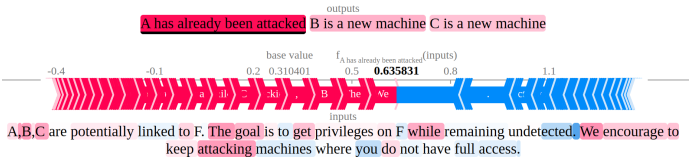

Fig. 4. Shap value of the action ”A has already been attacked”

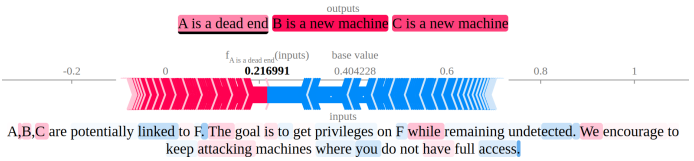

Fig. 5. Shap value of the action ”A is a dead end”

gains in confidence in the decision (compared to the previous context). With the rule given in the context (”We encourage to keep attacking machines where you do not have full access.”), the confidence in the selection of machine A increases. In the context 3, the word ”dead end” is used when the agent discovers a restriction of service. Thus, we can observe that the confidence about A1 is the lowest. We found that the use of particular words (”dead end”, ”arrested”, or ”stop”) in the observation of an action has a negative effect and reduces the probability of choosing that action. Figures 4 and 5 show the Shap value and confirm this effect. We observe that without the term ”dead end”, many keywords from the context are used (red colour) however when the sequence ”is a dead end” is given, no more words from the context is used. Thus, when our agent discovers a restriction, we include this word into the context in order to avoid that our agent gets blocked and therefore selects another machine. This behaviour is confirmed in the context 3, where our agent prefers to select the machine B since the machine A is a dead-end. In the context 4, the action B1 is selected because the machine A is totally corrupted. Thus, our agent selects a new machine such

as B or C. Finally, in the context 5, our agent discovered that the target machine is C, the confidence is then higher for the action C1.

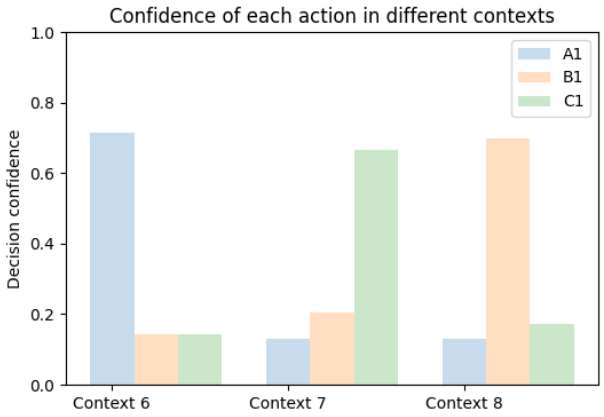

Fig. 6. Confidence in decision for different contexts when selecting actions

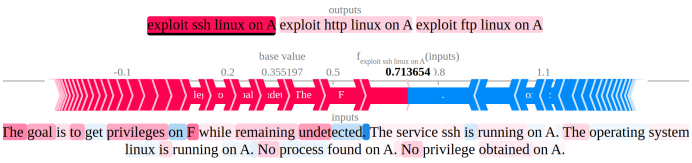

Fig. 7. Shap value of the action ”exploit ssh linux on A”

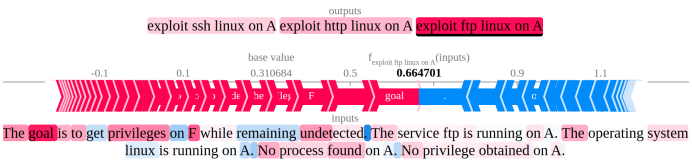

Fig. 8. Shap value of the action ”exploit ftp linux on A”

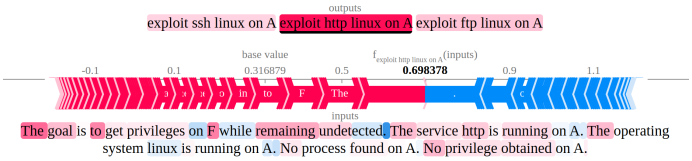

Fig. 9. Shap value of the action ”exploit http linux on A”

2) Services selection: In this section, we show how the words that correspond to the name of service impact the decision of our model. In order to illustrate this, we have created three other contexts called ”Context 6”, ”Context 7”, and ”Context 8” and three candidate labels per context. These new contexts and candidate labels are illustrated and described in Table VIII and IX. In the Figure 6, we can observe the probability of each action A1, B1, C1 for the different contexts. In the context 6, the service ssh is running on the machine. The action with the highest probability is A1, ”exploit ssh”, since the model knows that the associated service is running. In the other contexts, we can observe the same results. So if we change the name of the service running on the machine, the action exploiting this service will have a higher probability to be picked than the others. We also confirm that our model uses the name of services available in the context with the different Shap values present in the figures 7, 8, and 9. In the Figures 8 and 9, we observe that the word ”http” and ”ftp” are used to take a decision.

V. Discussion / Future works

In this paper, we have presented a more robust and adaptive approach based on a zero-shot classification that outperforms the other RL agents (DQN, PPO recurrent, and CLAP) in the reference NASim environment according to our new metric. We have made sure that our solution for building the environment is easily adaptive to any simulated network topology. We already have first conclusive results when exploiting the model in a realistic cyberrange environment. In order to improve the generalisation of the model, the next step is to test it in a more realistic and complex environment. We also want to improve our approach by using a more powerful AI architecture. Indeed, we plan on using Text2Text generation such as Llama [10], Huggingchat [7], FLAN [12], or Bloom [13] models instead of a zero-shot classifier with the BART model. We hope that using one of these models will reduce the amount of help we currently give in the observable environment. Doing so would also improve handling more complex pentesting challenges with more constraints, restrictions and services.

Our version of NASim and our naive evaluation metric will be available at the following URL: https://github.com/silicom-hub/zero-shot-pentesting-paper.

References

[1] Alessandro Confido, Evridiki V. Ntagiou, and Marcus Wallum. Reinforcing penetration testing using ai. In *2022 IEEE Aerospace Conference (AERO)*, pages 1–15, 2022.

[2] Naman Goyal, Jingfei Du, Myle Ott, Giri Anantharaman, and Alexis Conneau. Larger-scale transformers for multilingual masked language modeling. *CoRR*, abs/2105.00572, 2021.

[3] Zhenguo Hu, Razvan Beuran, and Yasuo Tan. Automated penetration testing using deep reinforcement learning. In *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 2–10, 2020.

[4] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *CoRR*, abs/1910.13461, 2019.

[5] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[6] Jonathon Schwartz and Hanna Kurniawati. Autonomous penetration testing using reinforcement learning. *CoRR*, abs/1905.05965, 2019.

[7] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface, 2023.

[8] Maxwell Standen, David Bowman, Son Hoang, Toby Richer, Martin Lucas, Richard Van Tassel, Phillip Vu, Mitchell Kiely, KC C., Natalie Konschnik, and Joshua Collyer. Cyber operations research gym. https://github.com/cage-challenge/CybORG, 2022.
[9] Maxwell Standen, Martin Lucas, David Bowman, Toby J. Richer, Junae Kim, and Damian Marriott. Cyborg: A gym for the development of autonomous cyber agents, 2021.
[10] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.
[11] Khuong Tran, Ashlesha Akella, Maxwell Standen, Junae Kim, David Bowman, Toby Richer, and Chin-Teng Lin. Deep hierarchical reinforcement agents for automated penetration testing, 2021.
[12] Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. Finetuned language models are zero-shot learners, 2022.
[13] BigScience Workshop, :, Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, and Daniel Hesslow. Bloom: A 176b-parameter open-access multilingual language model, 2023.
[14] Yizhou Yang and Xin Liu. Behaviour-diverse automatic penetration testing: A curiosity-driven multi-objective deep reinforcement learning approach, 2022.
[15] Wenpeng Yin, Jamaal Hay, and Dan Roth. Benchmarking zero-shot text classification: Datasets, evaluation and entailment approach. *CoRR*, abs/1909.00161, 2019.
[16] Shicheng Zhou, Jingju Liu, Dongdong Hou, Xiaofeng Zhong, and Yue Zhang. Autonomous penetration testing based on improved deep q-network. *Applied Sciences*, 11(19), 2021.

VI. Annexe

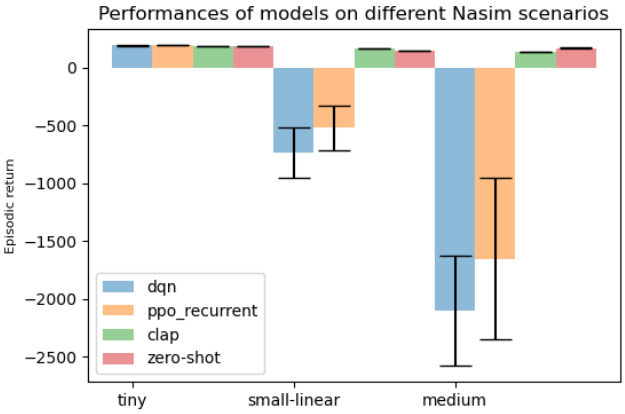

Fig. 10. Comparison of performances from RL agents (DQN, CLAP, PPO recurrent and zero-shot) trained and evaluated with the NaSim metric evaluation

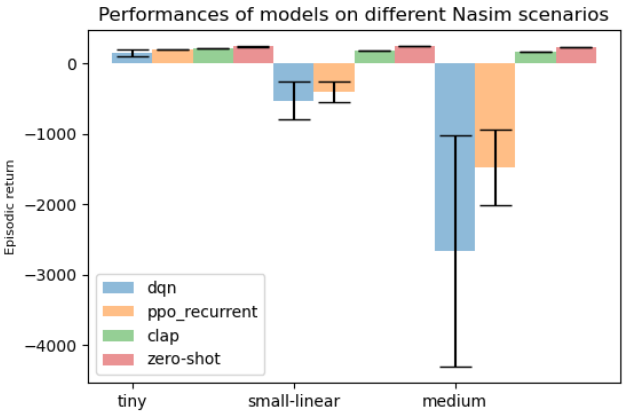

Fig. 11. Comparison of performances from RL agents (DQN, CLAP, PPO recurrent and zero-shot) trained with the NaSim metric and evaluate with the new metric evaluation

Context	Description
Context 1, 2, 3, 4, 5	A,B,C are potentially linked to F. The goal is to get privileges on F while remaining undetected.We encourage to keep attacking machines where you do not have full access.

TABLE VI
List of contexts for machine selection

Context	Candidate label	Description
Context 1	A1	A is a new machine
	B1	B is a new machine
	C1	C is a new machine
Context 2	A1	A has already been attacked
	B1	B is a new machine
	C1	C is a new machine
Context 3	A1	A is a dead end
	B1	B is a new machine
	C1	C is a new machine
Context 4	A1	A is totally corrupted
	B1	B is a new machine
	C1	C is a new machine
Context 5	A1	A is totally corrupted
	B1	B is a new machine
	C1	C is one of the target machine

TABLE VII
LIST OF CONTEXTS AND CANDIDATE LABELS FOR MACHINE SELECTION

Context	Description
Context 6	The goal is to get privileges on F while remaining undetected. The service ssh is running on A. The operating system linux is running on A. No process found on A. No privilege obtained on A.
Context 7	The goal is to get privileges on F while remaining undetected. The service ftp is running on A. The operating system linux is running on A. No process found on A. No privilege obtained on A.
Context 8	The goal is to get privileges on F while remaining undetected. The service http is running on A. The operating system linux is running on A. No process found on A. No privilege obtained on A.

TABLE VIII
LIST OF CONTEXT FOR MACHINE SELECTION

Context	Candidate label	Description
Context 6	A1	exploit ssh linux on A
	B1	exploit http linux on A
	C1	exploit ftp linux on 1
Context 7	A1	exploit ssh linux on A
	B1	exploit http linux on A
	C1	exploit ftp linux on A
Context 8	A1	exploit ssh linux on A
	B1	exploit http linux on A
	C1	exploit ftp linux on A

TABLE IX
LIST OF CONTEXT AND CANDIDATE LABEL FOR MACHINE SELECTION