IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received July 7, 2021, accepted July 23, 2021, date of publication July 28, 2021, date of current version August 6, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3100706

Towards Autonomous Defense of SDN Networks Using MuZero Based Intelligent Agents

JON GABIRONDO-LÓPEZ[1,2], JON EGAÑA[1], JOSE MIGUEL-ALONSO[3], (Member, IEEE), AND RAUL ORDUNA URRUTIA[1]
[1]Vicomtech Foundation, Basque Research and Technology Alliance (BRTA), 20009 Donostia-San Sebastián, Spain
[2]Faculty of Informatics, University of the Basque Country (UPV/EHU), 20018 Donostia-San Sebastián, Spain
[3]Department of Computer Architecture and Technology, University of the Basque Country (UPV/EHU), 20018 Donostia-San Sebastián, Spain

Corresponding author: Jon Gabirondo-López (jon.gabirondol@ehu.eus)

This work was supported in part by the Spanish Centre for the Development of Industrial Technology (CDTI) through the Project ÉGIDA—RED DE EXCELENCIA EN TECNOLOGIAS DE SEGURIDAD Y PRIVACIDAD under Grant CER20191012, in part by the Spanish Ministry of Science and Innovation under Grant PID2019-104966GB-I00, in part by the Basque Business Development Agency (SPRI)-Basque Country Government ELKARTEK Program through the projects TRUSTIND under Grant KK-2020/00054 and 3KIA under Grant KK-2020/00049, and in part by the Basque Country Program of Grants for Research Groups under Grant IT-1244-19.

ABSTRACT The Software Defined Networking (SDN) paradigm enables the development of systems that centrally monitor and manage network traffic, providing support for the deployment of machine learning-based systems that automatically detect and mitigate network intrusions. This paper presents an intelligent system capable of deciding which countermeasures to take in order to mitigate an intrusion in a software defined network. The interaction between the intruder and the defender is posed as a Markov game and MuZero algorithm is used to train the model through self-play. Once trained, the model is integrated with an SDN controller, so that it is able to apply the countermeasures of the game in a real network. To measure the performance of the model, attackers and defenders with different training steps have been confronted and the scores obtained by each of them, the duration of the games and the ratio of games won have been collected. The results show that the defender is capable of deciding which measures minimize the impact of the intrusion, isolating the attacker and preventing it from compromising key machines in the network.

INDEX TERMS Automated response, cybersecurity, intelligent agents, Markov games, MuZero, network security, OpenFlow, software defined networking.

I. INTRODUCTION

The number of Internet users has grown considerably in recent years and more and more services—from e-commerce to banking—are provided over the Internet. Consequently, not only the number, but also the severity of cyberattacks on organizations and businesses has been increasing, causing millions of dollars in losses [1].

Enterprise security systems have traditionally been designed and implemented manually by expert personnel and the response to attacks or intrusions has also been carried out by those technicians. The new network infrastructure paradigm introduced by Software Defined Networking (SDN) has enabled the development of automatic attack detection and mitigation systems based on machine learning techniques [2]–[6]. Compared to conventional detection systems or human-driven response strategies, these systems can detect attacks faster and more accurately, and even implement countermeasures autonomously and automatically, minimizing the reaction and response time to an attack and thus reducing the damage caused in the network.

Additionally, the development of reinforcement learning algorithms capable of outperforming human marks in board games—such as chess or Go—has fostered the idea of approaching network intrusion as if it were a Markov game, using these algorithms to train intelligent agents that can autonomously find security strategies to mitigate intrusions [7].

This paper presents an intelligent system which can minimize the impact caused by an intrusion in an SDN network by autonomously choosing and executing adequate countermeasures. The problem is posed as a partially observed Markov game in which the attacker (the intruder) tries to compromise a critical machine and the defender (an automatic security

The associate editor coordinating the review of this manuscript and approving it for publication was Nabil Benamar.

agent that manages the infrastructure) tries to reduce the impact of the attack. The game has been designed taking into account that it must be able to represent the state of the nodes of a real network and that the actions of the defender in the game must be implementable by an SDN controller. The model has been trained using the MuZero model-based reinforcement learning algorithm [8].

The major contributions of this paper are:

- The design of a Markov game suitable to represent attack attempts to a computer network in which nodes have vulnerabilities.
- The implementation of the necessary countermeasures by means of an SDN controller and OpenFlow-enabled switches.
- The implementation of a virtual environment in which the countermeasures chosen by the defender are carried out autonomously in an emulated SDN network.

The rest of this paper is organized as follows: Section II introduces the context of SDN and reinforcement learning and presents the state of the art in both fields. Section III presents the proposal made in this work, explaining how the game works and its integration with an SDN network. Section IV shows the information regarding the training of the model and the results obtained. Finally, Section V evaluates these results and highlights some possible lines of future work.

II. BACKGROUND AND STATE OF THE ART

A. BACKGROUND

1) SOFTWARE DEFINED NETWORKING

In recent years, the applications and services offered on the Internet have become increasingly complex and demanding. This has highlighted the need for a paradigm shift in the world of network infrastructures, as conventional networks lack the dynamism and adaptability required by the new platforms [9].

Conventional networks are composed of elements (such as routers and switches) usually treated as *black boxes* that rely on limited or manufacturer-specific control interfaces. These devices are typically configured individually, and traffic management decisions are made directly at the device level, intermingling the *control plane* responsible for making such decisions with the *data plane* composed of the network elements [10], [11]. The lack of independence between these two planes makes it extremely difficult to dynamically adapt the network to the needs of the applications deployed, or to cope with certain type of events. This problem is one of the main causes of the "ossification of the Internet": the development of new protocols and infrastructures has been severely hampered by the very architecture of network elements [10], [12].

Software defined networking proposes systems in which the control plane and the data plane are completely decoupled, allowing centralized configuration of the infrastructures, as shown in Fig. 1. Although there are different

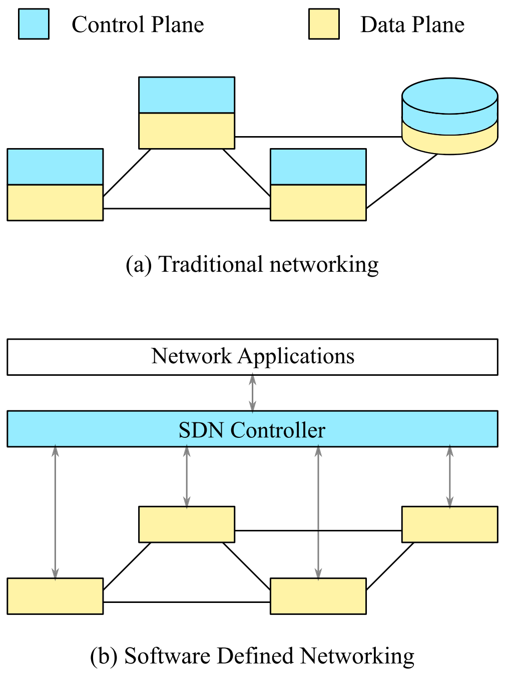

FIGURE 1. Structural differences between (a) a conventional network and (b) a SDN.

SDN architectures, this paper only considers those based on the OpenFlow protocol [12] in the communication between the control plane and the data plane, as it is a widely used protocol endorsed by the Open Networking Foundation (ONF)—the non-profit organization dedicated to the development and standardization of SDN networks [11]. The elements that constitute the network (known as OpenFlow switches) are only responsible for forwarding traffic, whereas decisions are made by the network controller. The controller installs *flow tables* in the switches using the OpenFlow protocol. The tables consist of *flows* that determine how packets that meet certain criteria are processed and forwarded.

A flow is composed of different elements: the *Match* fields are the conditions that the incoming packet has to meet (such as the source IP address or the incoming port, for example) to execute the instructions defined in the *Instructions* field. These instructions may include blocking all traffic coming from some port, forwarding the matching traffic through a specific output port or forwarding some packets to the controller, for example. When an incoming packet matches with more than one flow, the one with the highest value in the *Priority* field is executed. The *Counters* field collects the number of packets processed by that flow, the *Timeout* field defines how much time must pass without inputs for a flow to be dropped and the *Cookie* field is just an identifier set by the controller [13]. When a packet that does not match any flow is received, the instructions defined in the special *table-miss* flow are executed, which may include actions such as dropping the packet or sending it to the controller for further analysis.

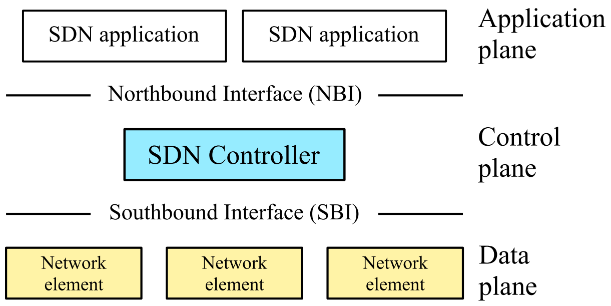

FIGURE 2. Diagram of the components of a SDN and the interfaces between them.

Fig. 2 shows the structure of this type of SDN networks, which consists of three main groups of components (planes) and two interfaces that allow communication between them. There are more complex structures that include several controllers and more interfaces, but in this paper, for the sake of simplicity, we only consider the case of a single controller. The controller uses the Southbound Interface (SBI) to communicate with the network devices. In this way, the controller manages all packet processing by installing flows in the switches. The controller can also collect information and statistics about the data plane. The SBI is vendor-independent and the most widely used is the aforementioned OpenFlow [14].

The controller also communicates with the *application plane* through a Northbound Interface (NBI). The application plane consists of several SDN applications that implement traffic control and management strategies, such as load balancing, packet filtering (firewalling), traffic monitoring, etc. Therefore, the NBI allows applications to process the data plane information received by the controller, perform actions at a high level and have the controller execute them on the infrastructure. There are no standardized NBIs, they are controller-specific. However, they are typically built using REST APIs [15].

An important property of a SDN is its ability to react to different events. The controller can perform an initial, proactive configuration of the network. However, this configuration can be dynamically modified (by adding or removing flows), adapting it to the traffic observed in the network, whether harmless or dangerous. SDN networks have therefore been used on many occasions to design systems capable of detecting and responding dynamically to cyberattacks [6], [16], [17].

2) REINFORCEMENT LEARNING AND MARKOV GAMES

Reinforcement Learning (RL) is an area of machine learning that studies how an agent learns to make decisions by following a trial-and-error strategy [18], [19]. The main elements of the RL models are shown in Fig. 3. The *agent* is the subject of the training and the one who has to learn to perform decisions. The *environment* represents the world with which the agent interacts [20]. In each interaction t, the agent receives an observation $o_t \in \mathcal{O}$ of the state of the environment $s_t \in \mathcal{S}$,

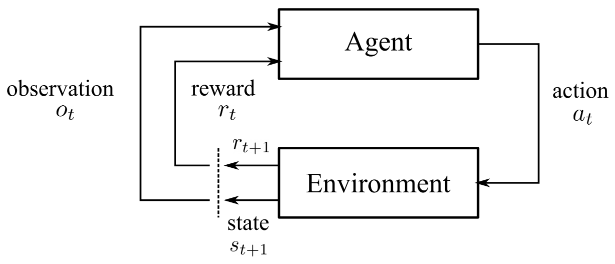

FIGURE 3. Summary of the main elements of reinforcement learning problems and the interactions among them.

where $\mathcal{O}$ is the set of observations of the ensemble of possible states $\mathcal{S}$. Based on the observation it decides which action $a_t \in \mathcal{A}(s_t)$ to take, where $\mathcal{A}(s_t)$ is the set of possible actions in state s_t. Then, the environment changes in response to the agent's action or independently. Once the action is executed, the agent receives a reward r_{t+1} and a new observation of the environment, which allows it to evaluate the effect of the action just performed, as well as to perform a new step.

It is worth noting the difference between *the state* and *an observation* of the environment. The state represents all the information of the environment and thus defines it completely. An observation, however, is a view that usually contains only a part of the information about the state. In the particular case where the observation includes all the information about the state, the environment is *fully observed*; otherwise, the environment is *partially observed*. In many cases, the term "state" is used to actually refer to "observation", and the symbol s_t is used instead of o_t. In this paper we have chosen to use an explicit expression of the observation, for two main reasons: the first is that a main characteristic of the model we have implemented is that it is based on a partially observed environment; the second is that the authors of the algorithm used to train the model use this notation in their article [8]. A main piece of a RL model is the *policy*, i.e., the strategy that determines what action the agent should take given an observation. This function is updated as the model is trained. In general, the policy is the probability that the agent will take an action a_t based on the observation o_t. The policy π usually depends on a set of parameters θ, so its complete representation would be $\pi_\theta(a_t \mid o_t)$.

A succession of states visited and actions taken, $\tau = (s_0, a_0, s_1, a_1, \ldots)$, is often referred to as an *episode*. The agent's main goal is to maximize the sum of all rewards obtained in an episode. This sum of rewards is known as *return*. A strategy called *discounted return* is often used, which makes rewards earned several steps back worth less than rewards earned closer in time. This return is defined as

$$R(\tau) = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \quad (1)$$

where γ is a parameter, $0 \leq \gamma \leq 1$, known as *discount rate*.

Beyond the intuition established so far, those processes studied in RL are *Markov Decision Processes* (MDP) which are defined by a six-component tuple as shown in (2) [21].

$$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}_{ss'}, \mathcal{R}^a_{ss'}, \gamma, \rho_0 \rangle \tag{2}$$

being

$$\mathcal{P}^a_{ss'} = \Pr\left[s_{t+1} = s' \mid s_t = s, a_t = a\right] \tag{3}$$
$$\mathcal{R}^a_{ss'} = \mathbb{E}\left[r_{t+1} \mid a_t = a, s_t = s, s_{t+1} = s'\right] \tag{4}$$

where $\mathcal{P}^a_{ss'}$ is the probability of reaching state s' by applying an action a in state s, and $\mathcal{R}^a_{ss'}$ is the expected reward after that transition. The term $\rho_0 : \mathcal{S} \mapsto [0, 1]$ is the distribution of the initial state [7].

The MDPs are so named because the transition from state s_t to state s_{t+1} satisfies the Markov property (5): it only depends on the mentioned states and not on the rest of the previous states [20].

$$\Pr\left[s_{t+1} \mid s_t\right] = \Pr\left[s_{t+1} \mid s_1, \ldots, s_t\right] \tag{5}$$

As mentioned in the introduction, the model we describe in this paper is a Markov game based on a *Partially Observed Markov Decision Process* (POMDP), in which the agent makes decisions based on its (partial) observation of the system and not directly on the state. At a step t, the observation $o_t \in \mathcal{O}$ is related to the state $s_t \in \mathcal{S}$ by the function $\mathcal{Z}$, so a POMDP is defined as follows:

$$\mathcal{M}_{\mathcal{P}} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}_{ss'}, \mathcal{R}^a_{ss'}, \gamma, \rho_0, \mathcal{O}, \mathcal{Z} \rangle \tag{6}$$

where $\mathcal{O}$ is the set of observations of the system and $\mathcal{Z} = \Pr\left[o \mid s\right]$ is the probability of observing o in state s.

Bringing the two fields together, the problem posed in RL can be understood as the search for the optimal policy π^* that maximizes the expected value $\mathbb{E}$ of the sum of rewards of an MDP for a maximum of T steps:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r_{t+1}\right] \tag{7}$$

In RL algorithms, value functions are defined, which calculate the expected return of actions taken from an initial state and following a certain policy. Formally, in the case of MDPs, the function $V_\pi(s)$ called *state-value function for policy* π is defined as

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi\left[R \mid s_t = s\right] \\ &= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right] \end{aligned} \tag{8}$$

and represents the expected return when the agent follows a policy π starting from a state s.

Analogously, the value of taking an action a in a state s, the *action-value function for policy* π, is also defined as

$$\begin{aligned} Q_\pi(s, a) &= \mathbb{E}_\pi\left[R \mid s_t = s, a_t = a\right] \\ &= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right] \end{aligned} \tag{9}$$

which represents the expected return obtained by following a policy π after performing action a in state s.

The *Bellman equations* set out in (10) and (11) show that the value of a state or state-action pair is the expected return obtained by following the optimal policy π^* defined in (7) [22].

$$V^*(s) = \max_a \mathbb{E}\left[r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a\right] \tag{10}$$

$$Q^*(s, a) = \mathbb{E}\left[r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') | s_t = s, a_t = a\right] \tag{11}$$

Recall that the ultimate goal of a RL algorithm is to find the optimal policy π^*. For finite MDPs, (10) has a unique solution, so once V^* has been calculated it is relatively simple to obtain the optimal policy, since if this function is used to evaluate the short-term actions (evaluating the state s arrived at after performing a) the policy followed when taking the best option at each step is the optimal long-term policy [20].

Similarly, using Q^*, the agent no longer has to search for the new state s that maximizes $V(s)$, but only has to search for the action a that results in a higher $Q(a)$. This allows the agent to make decisions without having to know all the consecutive states, that is, without having to know the dynamics of the environment with which it interacts.

Estimating these value functions is a fundamental part of *model-free* algorithms, which seek to find the optimal policy without having a model of the environment, as the *Q-learning* [23] algorithm does, for example. On the other hand, *model-based* algorithms also use such functions to construct models that have the same value functions as the original environment and are therefore equivalent. The algorithm MuZero [8] is one of the most recent examples of model-based algorithms.

Finally, it only remains to mention that decision processes involving multiple agents are studied within game theory [24], [25]. These agents interact with the environment simultaneously or in turn and each one obtains a corresponding payoff, thus refining their policies. When designing a game, the following factors must be determined, which also serve to classify them [26]:

- **Zero-sum:** Whether the sum of all players' rewards is 0 or not. In two-player games, rewards sum to 0 if both players are strictly competing against each other.
- **Information:** Whether the game state is fully or partially observable by the players.
- **Determinism:** Whether the outcome of the game depends to some extent on luck.
- **Sequential:** Whether the agents interact sequentially or simultaneously.
- **Discrete:** Whether actions are implemented in real time or not.

Specifically, Markov games are the theoretical context of *multi-agent* RL algorithms [27]. A Markov game $\mathcal{M}_G$ with N agents is defined by a tuple similar to the one given in (5):

$$\mathcal{M}_G = \langle \mathcal{S}, \mathcal{A}_1, \ldots, \mathcal{A}_N, \mathcal{T}, \mathcal{R}_1, \ldots, \mathcal{R}_N, \gamma, \rho_0 \rangle. \tag{12}$$

Compared to an MDP, the Markov game defined by (12) presents a list of sets of possible actions to be performed by the N agents in which to an agent $i \in [1, N]$ corresponds its set $\mathcal{A}_i$ instead of the unique set $\mathcal{A}$. The transition function $\mathcal{T}$ operates on all possible combinations between the set of states and the combined action space of all agents: $\mathcal{T} : \mathcal{S} \times \mathcal{A}_1 \times \mathcal{A}_2 \ldots \times \mathcal{A}_N \mapsto \mathcal{S}$. Similarly, the functions $\mathcal{R}_i$ define the rewards obtained by the agents: $\mathcal{R}_i : \mathcal{S} \times \mathcal{A}_i \mapsto \mathbb{R}$. In the case of a game based on a partially observed process, the definition (12) also includes the lists of sets of observations $\mathcal{O}_1, \ldots, \mathcal{O}_N$ and of observation functions $\mathcal{Z}_1, \ldots, \mathcal{Z}_N$ [7].

B. STATE OF THE ART

1) INTRUSION DETECTION AND PREVENTION SYSTEMS

Intrusion detection systems (IDS) attempt to identify malicious activity occurring on a network (network IDS) or on a computer (host IDS) by capturing and analyzing different sources of information. A network IDS captures and analyses the packets traversing the network (both protocol headers and content), as well as aggregate measurements on traffic: sessions, IP addresses involved, traffic volumes, etc. They are usually guided by a set of rules against which the captured data is compared. If there is a match, an alert is issued: the IDS has detected suspicious activity. Intrusion detection and prevention systems (IDPS) go one step further: they are not only capable of detecting an intrusion attempt, but also of reacting to reduce or completely prevent the malicious effects by executing some countermeasures. To this end, an IDPS is able to perform network modifications, for example by automatically adding a rule in the corporate firewall [28].

SDNs are particularly suitable for IDPS deployment. They can be deployed as control applications, capable of collecting information from the network with the help of the controller. The detection subsystem will then search the collected data for signs of intrusion. If dangerous activities are detected, the IDPS intelligence will choose the best course of action, consisting of a series of countermeasures that affect the flow tables of the network devices. The necessary changes will be implemented through requests to the controller.

The work presented here is intended to be only one piece of a complete IDPS. We assume that there is a working IDS that tells our intelligent agent what type of attack has been detected. Our agent will choose the best countermeasures and ask the controller to apply them. This is the part we focus on. The design and implementation of a good network IDS goes beyond the scope of our work. The interested reader can find relevant and up-to-date information on network IDS in [29]–[32].

2) AUTONOMOUS INTRUSION MITIGATION SYSTEMS IN SDNs

As stated before, the abstraction, flexibility and programmability of infrastructures based on the SDN paradigm have enabled the development of attack detection and automatic countermeasure deployment systems [5], [33]. Focusing on the defense activities, several intelligent systems capable of deciding which countermeasures to take have been proposed. One of the most complete systems is NICE (Network Intrusion detection and Countermeasure sElection in virtual network systems) [6], which integrates the detection of infected virtual machines in cloud computing environments with the automatic deployment of optimal countermeasures, which include patching the software of an attacked machine or quarantining a suspicious node. The entire infrastructure proposed in NICE is based on SDN elements using the OpenFlow protocol. Each machine in the network has a vulnerability registered in the Common Vulnerabilities and Exposures (CVE) list [34], and the attack is represented by an attack graph where each node represents the previous state or consequence of the exploit on one of the machines. The selection of the response to an attack is made taking into account the intrusiveness and cost of the countermeasure, so as to minimize the impact of the response itself. The SnortFlow [17] system follows the guidelines set by NICE, but all proposed countermeasures are based solely on actions taken on the network itself (such as redirecting traffic or blocking a port) and are implemented via an OpenFlow controller.

Apart from proposals focusing on a practical aspect of countermeasure implementation, works such as [7] propose to use multi-agent Markov games to search for defense strategies against cyberattacks. Specifically, the game is set up as a partially observed game and the authors study different scenarios in which the abilities of the attacker and defender vary. In their game, the attacker must move through a graph to reach a target machine, simulating a problem similar to those posed in the Cyber Ranges [35]. In that work, the model-free algorithms PPO [36] and REINFORCE [37] are used to train the agent.

3) REINFORCEMENT LEARNING ALGORITHMS AND THE MuZero ALGORITHM

A classic way of evaluating reinforcement learning algorithms is to pit them against games such as chess or Go in an attempt to beat human scores and obtain superhuman results. One of the first milestones in this field, apart from specialized computers for winning at chess or shogi [38], was the AlphaGo algorithm developed by the company DeepMind, which managed to beat a professional Go player for the first time [39]. A modification of that algorithm called AlphaGo Zero [40] achieved superhuman results playing Go and led to its successor AlphaZero, which managed to beat world champions in chess, shogi and Go with only 24 hours of training in each case [41]. AlphaZero was trained using 5000 first-generation TPUs to generate self-play games and 64 second-generation TPUs to train the neural networks.

Model-based algorithms have repeatedly outperformed humans in classic games such as checkers, chess, Go or poker, as they are able to develop a long-term strategy.

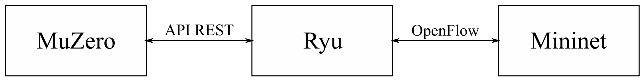

FIGURE 4. Diagram of the proposal made in this work, in which the trained model, the network controller and the simulation environment are completely independent.

However, those algorithms have tended to perform poorly when confronted with complex environments like the Atari 2600 computer games. Model-free algorithms obtain better results in these environments [42]–[44], but perform much worse than model-based algorithms in games such as those mentioned above, which require precise and sophisticated lookahead [8].

The MuZero algorithm introduced by DeepMind in 2020 (a model-based algorithm) completely changed the state of the art. It obtains performances comparable to those of model-free algorithms in visually complex environments, while maintaining superhuman results in precision planning tasks such as classic board games, as previous model-based algorithms [8].

III. PROPOSED ARCHITECTURE

In this work we propose to use the MuZero algorithm to train a model that is able to decide which countermeasures to implement when an attacker intrudes an SDN network, with the intention of mitigating the attack and minimizing the number of compromised machines. The interaction between the intruder and the defender has been modeled as a stochastic, partially observed, zero-sum Markov game, in which both agents perform actions sequentially. Furthermore, the model has been integrated into a realistic SDN network emulation environment based on Mininet [45]. The (virtual) switches deployed within the Mininet environment support the OpenFlow specification and accept control from an external controller—we have used the Ryu controller [46]. Fig. 4 summarizes the three blocks developed for this work.

Below we present the programs and libraries used for the implementation of our proposal, the decisions related to the game design and the virtual network environment implemented. The hardware used to train and test the model is described in Section IV-C.

A. DESIGN OF THE GAME

This section explains the different elements designed to obtain a partially observed Markov game representing the interaction between the attacker and the defender.

1) AGENTS AND THE ENVIRONMENT

The game is set up as a match between two players, the attacker and the defender, in which the attacker tries to compromise key targets in a network and the defender tries to mitigate the intrusion.

Although the MuZero algorithm was designed to deal with board games or video games, in this case the environment to be simulated is a network of interconnected computers, so we have chosen to use a graph to represent it. For practical purposes, MuZero can be used to train agents in any environment represented by a vector or a matrix. The translation from the network graph to a matrix suitable for MuZero is explained in Section III-A2.

In this game, the attacked network is composed of N vulnerable nodes, of which m are part of a *honeynet*. A honeynet is an isolated part of the network where machines (called *honeypots*) are used as traps for attackers [47]. These machines do not provide actual services, but are vulnerable to attacks, causing attackers to waste time and resources exploring while allowing defenders to obtain intruder-related information [48]. This structure is similar to the one proposed by the Science DMZ [49].

Inspired by the NICE project, each node in the network (also known as *host*) is assigned a Base Score (BS), an Exploitability, an Impact, and a Scope. The National Vulnerability Database (NVD) [50] uses those factors to quantitatively assess a vulnerability, which represents a weakness in software and hardware components of a system that, when exploited, negatively affects its confidentiality, integrity, or availability. In a real setup where the IDS would detect the intrusion, those factors would be computed using the equations defined by the version 3.1 of the Common Vulnerability Scoring System (CVSS) [51]. According to that standard, the value of the BS depends on the Impact sub-score (ISS), Exploitability, and Impact. The impact sub-score is defined as

$$ISS = 1 - [(1 - C) \times (1 - I) \times (1 - A)] \tag{13}$$

where C is the Confidentiality Impact, I the Integrity Impact and A the Availability Impact. Those metrics measure the impact to the confidentiality, integrity and availability of the information resources caused by the exploitation of the vulnerability of a software component.

The Exploitability of a host depends on the Attack Vector (AV), the Attack Complexity (AC), Privileges Required (PR) and User Interaction (UI) according to (14). Those metrics represent the properties of the vulnerability that lead to a successful attack. The Attack Vector increases with the physical and logical distance between the attacker and a successfully compromised component. The Attack Complexity represents the conditions that are beyond the attacker's control and that are necessary to exploit the component. The Privileges Required metric describes the privileges that the attacker must have before exploiting the vulnerable component and the User Interaction represents if the activity of another user other than the attacker is required to successfully compromise the vulnerability.

$$\text{Exploitability} = 8.22 \times AV \times AC \times PR \times UI. \tag{14}$$

The Impact of the vulnerability depends on its Scope (S). This factor determines whether the affected host is the one with the vulnerability (Unchanged) or a different one

(Changed). In the case where the Scope is Changed, the Impact is computed as

$$\text{Impact} = 6.42 \times ISS \tag{15}$$

whereas if it is Unchanged,

$$\text{Impact}=7.52\times(ISS-0.029)-3.25\times(ISS-0.02)^{15}. \tag{16}$$

Finally, BS is calculated differently depending on the Impact and Scope. If Impact $= 0$, $BS = 0$. If Impact > 0, BS is computed as

$$\text{roundup}\left(\min\left[1.08\times(\text{Impact}+\text{Exploitability}), 10\right]\right) \tag{17}$$

if Scope is Changed, and as

$$\text{roundup}\left(\min\left[(\text{Impact}+\text{Exploitability}), 10\right]\right) \tag{18}$$

if Scope is Unchanged. The roundup function returns the smallest number, specified to one decimal place, that is equal to or greater than its input.

In terms of the game, the Scope of a vulnerability determines what actions the attacker can take if he succeeds in exploiting it: if the Scope is Unchanged, the attacker could only read files from the target machine, whereas if the Scope is Changed, the attacker could scan and exploit the machines to which the compromised host is connected. The design decisions related with the Base Score, the Exploitability and the Impact will be discussed in Section III-A3.

The initial network therefore consists of $N - m$ hosts connected to each other forming a fully connected logical network and m hosts isolated from the main network but connected to each other (see Table 1 and Fig. 6 in Section IV-B). Each node is assigned a vulnerability and it can be in different states—normal (state 0), scanned (state 1) and attacked (state 2)—allowing only the transitions $0 \rightarrow 1 \rightarrow 2$. Machines cannot revert to a previous state, so that it is guaranteed that there is a reduced number of possible actions. At the start of the game one of the hosts on the main network is set to *flag* (state -1) and the attacker's main objective will be to reach that host; another host with Scope "Changed" is set to state 2, allowing the attack to start from there.

In order to illustrate the explanations that follow, in the rest of this paper we will represent the state of the network using graphs, since they are also used to show the state of the game. Formally, a graph $G = (V, E)$ is an ordered pair where V is a set of nodes or vertices and E is a set of edges such that $E \subseteq \{\{x, y\} \mid x, y \in V \;\wedge\; x \neq y\}$. In our representation of the system, the nodes in the graph represent the hosts, and the edges show that traffic between two hosts is not blocked (i.e. there is connectivity between them). As hosts can be in different states, a color coding has been chosen to represent them in the figures (see Fig. 5). In this coding, those machines in its initial state are represented by white background nodes, the explored ones by yellow nodes and the attacked ones by the purple nodes. A node with a dashed-dotted border represents the flag. The label placed inside each

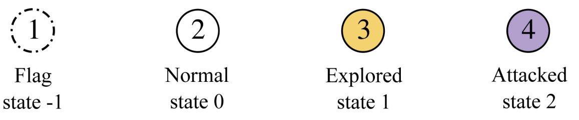

FIGURE 5. Encoding used to represent the states in which the nodes can be.

TABLE 1. Example of a game environment. The red colored row represents the attacked node and the green colored one is the flag.

Node	BS	Expl.	Imp.	Scope	Neighbors
1	9.9	6.0	3.1	Changed	2, 3, 4, 5, 6, 7, 8
2	8.3	5.5	2.8	Unchanged	1, 3, 4, 5, 6, 7, 8
3	6.5	2.5	3.9	Unchanged	1, 2, 4, 5, 6, 7, 8
4	7.9	4.2	2.5	Changed	1, 2, 3, 5, 6, 7, 8
5	6.5	2.5	3.9	Unchanged	1, 2, 3, 4, 6, 7, 8
6	8.3	5.5	2.8	Unchanged	1, 2, 3, 4, 5, 7, 8
7	6.8	4.2	2.5	Unchanged	1, 2, 3, 4, 5, 6, 8
8	7.9	4.7	2.5	Changed	1, 2, 3, 4, 5, 6, 7
9	6.5	2.5	3.9	Unchanged	10, 11, 12
10	10.0	6.0	3.9	Changed	9, 11, 12
11	8.3	5.5	2.8	Unchanged	9, 10, 12
12	7.9	4.2	2.5	Changed	9, 10, 11

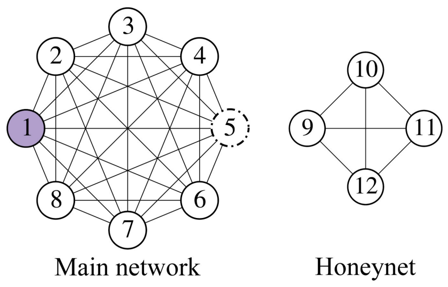

FIGURE 6. Graphical representation of the environment described by Table 1.

node is just an identifier, and has no relevance from the point of view of the game.

As an example, Table 1 defines a possible environment consisting of $N = 12$ nodes (with $m = 4$ nodes forming a honeynet), and Fig. 6 shows the graph representation of the network. It should be noted that these are reachability graphs that do not represent the physical topology of the network (consisting of switches, hosts and Ethernet links), since an edge between two nodes does not represent a physical link, but means that they can exchange packets. Note also that switches are not represented in these graphs.

2) THE OBSERVATIONS

The evolution of the intrusion and its mitigation is modeled as a partially observed decision process, since neither the attacker nor the defender has full information of the state of the network. To represent this fact we have used three different graphs: the actual or general graph $G_G = (V_G, E_G)$ with the complete state of the network, the attacker's graph $G_A = (V_A, E_A)$ with the partial view that the attacker has, and the defender's graph $G_D = (V_D, E_D)$ that is also a partial view.

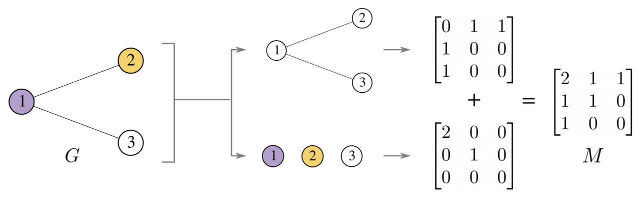

FIGURE 7. Calculation of the matrix *M* representing the observed state of the graph *G*.

Remember that we need a matrix representation of a game board, so we need to translate the above graphs into equivalent matrices. Although the adjacency matrix representation of a graph makes it possible to reflect the existing (logical) connections in the network, it is not sufficient to describe it in its entirety, since it does not include the different states in which nodes can be. Taking advantage of the fact that the main diagonal of the adjacency matrix is null, if we represent the state (real or observed) of all nodes according to the coding described in Fig. 5 as a diagonal matrix, a complete board can be represented by adding the two matrices. Therefore, the graphs G_G, G_A and G_D are fully represented by the matrices M_G, M_A and M_D, respectively. Fig. 7 summarizes the derivation of a board matrix for a simple graph with three nodes.

The three network graphs representing the state of the game are different from each other; an action taken by a player (explained in Section III-A3) only has an effect on the general graph and on the graph of that player. Consequently, each player only has a partial observation of the game. Fig. 8 shows three graphs/matrices computed for the same state. The actual state can be described as follows: starting from the initial situation determined by Table 1, the attacker has attacked machine 4 and has explored machine 2. In reality, machine 1 is isolated from the rest (as seen in the graphs G_G and G_D) but the attacker has not yet noticed the change in connectivity between the nodes of the network. Similarly, the defender has detected that machine 1 has been attacked (and we can assume that it has taken some of the actions explained in Section III-A3) but does not notice that node 2 is in state 1 and node 4 is in state 2.

3) THE ACTIONS

The following conditions have been considered in order to design the actions that each player can execute:

- Only defensive actions that can be implemented using an SDN controller are considered, leaving out of the study options such as software upgrades or changes to the underlying physical topology—such as adding or removing a switch or changing the switch/port to which a host is connected.
- Defensive actions can only be applied against machines that have already been attacked, but network monitoring is allowed.
- Offensive actions can be only carried out against the hosts: both switches and the controller are invulnerable.

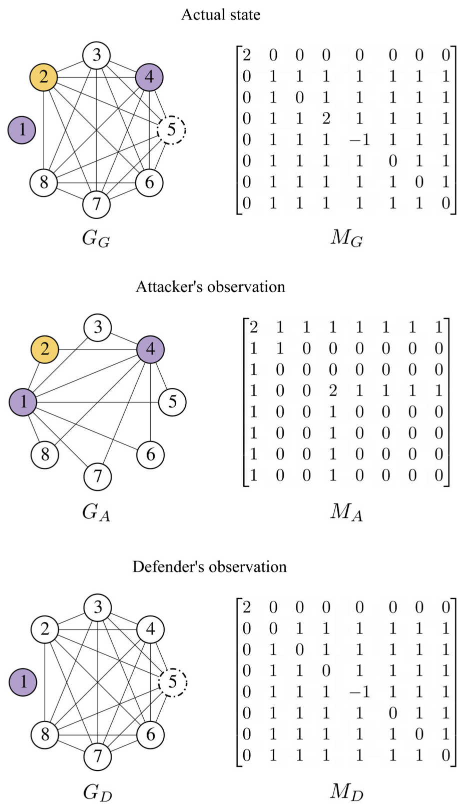

FIGURE 8. Example of the general state of a game and the observations seen by the defender and the attacker.

- The MuZero algorithm requires the board size to remain constant throughout the game, so the number of nodes cannot vary: new machines cannot be deployed or removed completely.

It should be noted that an agent's decision making is based on its observation of the game, not on the actual state. Therefore, *legal* actions that an agent can take theoretically may not be feasible in practice or may not have the expected effect due to differences between the agent's observation and the actual state. Moreover, as mentioned above, this is a stochastic game, so there are actions that have a probabilistic factor. For example, whether or not the attacker succeeds in exploiting a vulnerability in a machine depends directly on the characteristics of that vulnerability, so the agent will not always get the same result.

Actions available to the defender are:

- **Check status**
  The defender verifies whether the state of a node corresponds to that of its observation. If a node has been attacked, the probability of detecting it is proportional to the Base Score of the vulnerability of the attacked machine.

$$\Pr\left[o_D(n_i) = 2 \mid s(n_i) = 2, a = \text{check}\right] = BS/(10N) \tag{19}$$

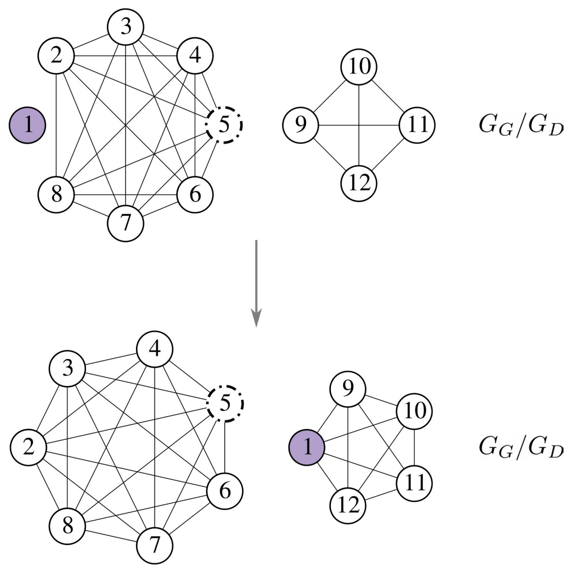

FIGURE 9. Main network and honeynet status before redirecting all traffic from node1 to honeynet nodes (up) and after (bottom).

where $o_D(n_i)$ and $s(n_i)$ are the defender's observation and the real state of the affected machine n_i and N is the total number of nodes in the network.

- **Isolate node**
  Deterministic action that completely isolates an exploited machine.
- **Send node to honeynet**
  Deterministic action that blocks all traffic between the affected machine and the main network and allows the node to communicate with the rest of the nodes in the honeynet (see Fig. 9). In our game, the goal of this action is to make the defender gain time to check the state of the rest of the network while the attacker scans and attacks the honeypots.
- **Move the flag**
  This action simulates the migration of critical services from one machine to another. The defender can try to perform this action if the objective host is not compromised according to its observation. Even though it is a deterministic action, if the machine has been attacked (the actual state of the node is 2 instead of the observed state 0), the action cannot be performed, but the defender can detect that the machine has been attacked with a probability equal to that used in the action *Check status* (19). Migration of services to multiple machines (flag splitting) is not considered.

Each countermeasure has been assigned a value that represents the total cost of carrying out that action. The countermeasures proposed in NICE [6] and those proposed in this paper are equivalent, so each action has been assigned the cost and intrusiveness of its equivalent measure. The action *Check status* has no equivalent in NICE, as it is not a countermeasure but a monitoring task, so it has been assigned a low cost and zero intrusiveness. The total cost of each measure, which takes into account the effect it has on the network and the

TABLE 2. Costs and intrusiveness of proposed countermeasures.

Defender's action	Intrusiveness	Cost	Total cost
Check status	0	1	1
Isolate node	4	2	6
Send node to honeynet	5	2	7
Move the flag	0	5	5

resources needed to implement it, is obtained by adding its cost and intrusiveness, see Table 2. These costs have been defined in order to make the defender mitigate attacks by following an optimal strategy, thus minimizing the resources used.

For the attacker, the following actions are available:

- **Explore the topology**
  The attacker agent obtains the list of machines connected to those already attacked with Scope "Changed", thus updating its observation.
- **Scan for vulnerabilities**
  This action scans the vulnerability of a machine from another machine with Scope "Changed". If the action is viable, the probability of detecting a vulnerability is its Exploitability/10, which is in the range of 0 to 1. Once scanned, node status changes from 0 to 1.
- **Attack vulnerability**
  Attacker attempts to exploit one of the vulnerable machines from another with Scope "Changed". The probability of compromising an explored machine is, again, proportional to the Exploitability of the vulnerability of that host: Exploitability/10. Once attacked, node status changes from 1 to 2.

Having presented all the actions, it is pertinent to return to the following idea: an attacker may decide to perform an action, but that does not mean that he succeeds in performing it. The pseudo-code presented in Algorithm 1 shows how an attacker action is executed. When carrying out an exploration or an attack, the intruder takes the set of neighboring nodes of the target machine $n_O \in V_A$ from the graph G_A. From that set it randomly chooses one of the nodes $n_A \in V_A$ with Scope "Changed" and tries to perform the action. If the edge between n_O and n_A does not exist in G_G, the attacker cannot use that connection to attack, so it removes the edge from its graph and selects another node. If the edge exists, a random number between 0 and 1 is generated and the Exploitability of the objective machine is divided by 10 to scale it to that range. Thus, the action succeeds if the random number is smaller than the scaled Exploitability, leading to a success probability of Exploitability/10.

4) THE OBJECTIVES

The attacker's main objective is to attack the flag machine and the defender's main objective is to prevent this from happening. To do this, both players take actions sequentially until the attacker reaches the flag (the attacker wins) or cannot take any more actions (the defender wins).

Algorithm 1 Progress of an Attack

```
action, objective_node ← select_action(observation)
neighbors ← get_neighbors(objective_node, $G_A$)
action_possible ← false
for neighbor $\in$ neighbors do
    if edge(objective_node, neighbor) $\in E_G$ then
        action_possible ← true
        break
    else
        remove_edge(objective_node, neighbor, $G_A$)
    end if
end for
if action_possible == true then
    exploitability ← get_exploitability(objective_node)
    random_number ← get_random_number(0, 1)
    if exploitability/10 $\geq$ random_number then
        apply_action(action, objective_node)
    end if
end if
```

5) THE REWARDS

The implementation of MuZero used poses two-player games as zero-sum games, automatically changing the sign of the opponent's payoff. Therefore, each player, in addition to receiving the reward it is entitled to, is also penalized with the payoff obtained by the opponent. The aim of each agent is to maximize its own reward while minimizing its opponent's, so secondary objectives beyond winning the game are specified by the design of the rewards.

In this work we have chosen to reward only the winner of the game at the end of it, giving a null reward to the rest of the actions. It must be taken into account that the game also ends after a certain number of moves, thus avoiding infinite games. Games that end by this criterion are considered a draw and neither player receives a reward. In order to train a defender capable of minimizing the effort required to mitigate the intrusion, the rewards shown in (20) have been implemented.

$$r_t = \begin{cases} TI, & \text{winner} = \text{attacker} \\ \max(0, MM - TI - TC), & \text{winner} = \text{defender} \\ 0, & \text{otherwise} \end{cases} \tag{20}$$

where TI is the total impact caused by the attacker, MM is the maximum number of moves allowed before the end of the game and TC is the total cost of the implemented countermeasures.

The total impact is the sum of the impact of the exploited vulnerabilities multiplied by 10, thus making it of the same order of magnitude as the total cost of the countermeasures and, consequently, of the reward obtained by the defender. The total cost is computed by adding up the costs of all the countermeasures implemented by the defender, using the values listed in Table 2. This reward system not only recompense the agent for trying to finish the game, but also positively evaluates the attacker for compromising as many nodes as possible, thus encouraging it to explore as much of the network as possible.

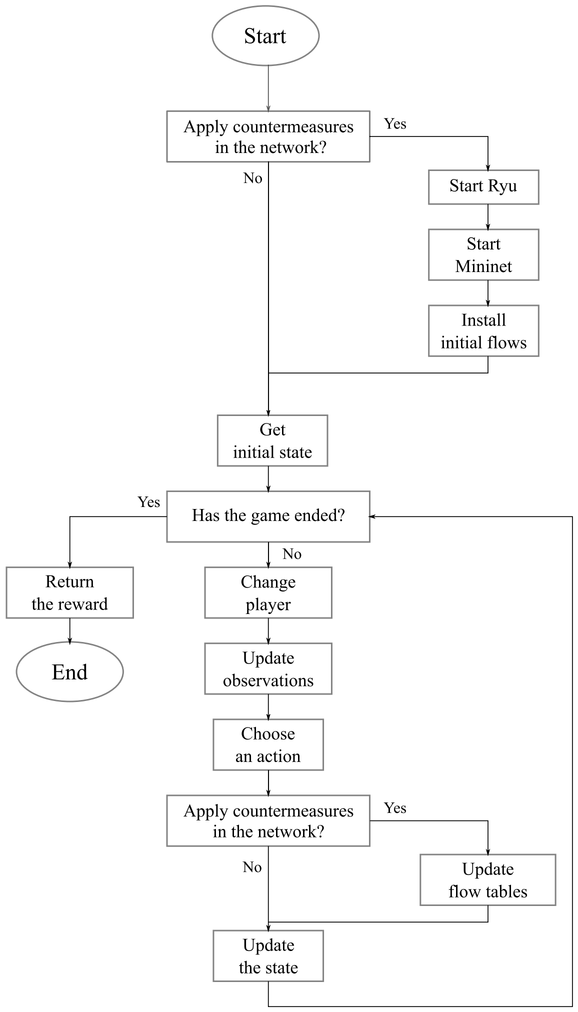

FIGURE 10. Diagram of the deployment and development of the game.

B. DEVELOPMENT OF THE GAME

Fig. 10 shows the deployment and development of the game. This sequence of actions is repeated every time MuZero starts a game. In this figure it can be seen how the network implemented in Mininet is used to mirror on it the countermeasures applied by the defender. When training the model, the game is played entirely on the boards, but, optionally, MuZero can interact directly with the SDN controller.

Keep in mind that the physical infrastructure doesn't change, but the connectivity does, and we take care to maintain the flows installed on the routers so that the connectivity matches the actual game board. We do checks to make sure the network and the game are in sync.

IV. EXPERIMENTAL DESIGN

This section presents the experiments performed and the results obtained for a relatively simple case study. In general terms, the experiments are based on defining a game environment, training the model using MuZero and playing games between models with different number of training steps. The number of moves required by each player to win the game—the episode length—, the points obtained as a reward by the winner and the percentage of games won by each player have been chosen as the metrics to measure the performance of the players.

A. SOFTWARE COMPONENTS

- **Mininet [45]:** used to define within a single guest machine virtual networks consisting of Ethernet links, OpenFlow switches and Linux hosts connected to the switches. The network topology, as well as switch, link and host parameters can be specified via command line or Python [52] scripting.
- **Ryu [46]:** OpenFlow controller. Ryu is an environment for programming OpenFlow-based SDN networks. It can be seamlessly integrated with virtual topologies defined with Mininet, although its main function is to control physical OpenFlow switches. The controller is programmed in Python (the language in which Ryu itself is written), and Ryu provides facilities to implement REST API-based northbound interfaces [15].
- **MuZero General [53]:** an implementation of MuZero based on the pseudo-code provided in the original [8] article and written in Python. This implementation provides a suitable environment for training custom games written in Python and evaluating the performance of the obtained model.

Consequently, this work has been developed entirely in Python [52].

B. UNDERLYING PHYSICAL INFRASTRUCTURE

Unlike [7], the second objective of our work is to implement a system that can actually perform countermeasures in an SDN network. As mentioned above, we use SDN networks emulated in Mininet and the Ryu controller. A game board is defined over the real network, and the actions taken by the defender correspond to countermeasures implemented (using Ryu) as reconfigurations of the network switches. This setup has two main objectives:

- Bridging the gap between the purely theoretical game and its application in real SDN networks.
- Be the basis for more complex implementations obtained as a result of replacing or extending the functionalities of any of the blocks—the game, the controller or the simulation environment—independently.

The configurations of the network topology and the Ethernet links between the hosts and switches are defined, together with the initial game state, in two JSON files: `topology.json` and `graph.json`. The first one determines the OpenFlow switches used, the number of hosts deployed and to which switch each of them is connected. The second file determines the characteristics of each host: the IP and MAC addresses, the hosts it can communicate with and the vulnerability it presents; it also specifies which nodes are initially attacked and which of them is the flag.

The basic operation of our networks is based on the behavior of traditional Ethernet switches, as defined in the Ryu documentation [54]: when a packet arrives at a switch and does not match any of flows of that switch, a copy of the packet is send to the controller. The controller then orders to (1) perform a "flood": a copy of the packet is sent through all ports except the one it was received on, and (2) install new flows on the switches in order to make forwarding straightforward in the future, without contacting the controller again. The result of this way of working is that the network allows all-to-all connectivity between hosts. At first it is not very efficient because the controller has to intervene to command the flooding, but the switches are gradually configured so that subsequent forwarding is done directly.

Once the network is built, additional flows are installed so that reachability is as determined by the configuration file. In parallel, the graphs corresponding to the game boards are created. It is worth mentioning that different network topologies can give rise to the same game graph, as these graphs do not depend on the network topology, but on the reachability.

Whereas the countermeasures have been implemented by Ryu, the attacks posed in the game should be understood as abstractions of the actions that an expert attacker would take and therefore are not mirrored in the network. The controller's execution of the actions chosen by the defender is shown below:

- **Isolate node**
  Flows are installed in the switches that block all packets coming from or going to the IP address of the isolated machine.
- **Send node to honeynet**
  A switch in a honeynet is randomly selected and the controller removes all flows that blocked traffic between the nodes in that honeynet and the target machine. In addition, flows are installed on the main network switches that block traffic to and from that machine.
- **Check status**
  Currently, this action has not been implemented in the controller. In the future, an agent could be added to study the state of the nodes in the network and detect intruders with a tool such a Security Information and Event Management system (SIEM) [55].
- **Move the flag**
  This action is not implemented by the controller, because it is not capable of changing the IP and MAC addresses of a host—that is how the flag could be moved without actually migrating the service from one computer to another. In future work, the controller will be used to redirect the traffic to the new host once the addresses have been changed.

In a real configuration with a complex SDN and several control applications operating on it, a mechanism should

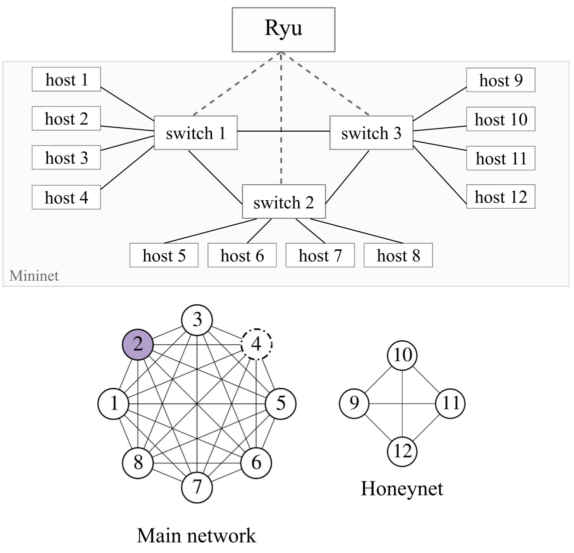

FIGURE 11. Diagrams of the network topology of the case study (top) and its logical configuration (bottom).

be put in place to avoid logical errors such as loops, black holes, rule overlapping, etc., like the one proposed in [56]. For simplicity we have not included it in our proof of concept.

C. HARDWARE USED

All experiments have been executed on a virtual machine with 20 vCPUs @2.2 GHz, 40 GB RAM and 2 NVIDIA Tesla M10 GPUs. The physical CPUs installed in the host server are two Intel Xeon Silver 4114. The hyperparameters related to MuZero can be found in Appendix A.

D. THE NETWORK AND THE GAME BOARD

We performed the experiments on a network with $N = 12$ hosts. Eight of them form the main network and are distributed over two switches, while the other four are part of a honeynet. The upper part of the Fig. 11 shows the actual network topology: solid lines represent Ethernet links and dashed lines are control connections between the controller and the switches, used to interchange OpenFlow messages. The lower part of the same figure represents the reachability network: with which other nodes each node can exchange traffic. The purple node is the first node attacked, and the node with dashed-dotted border is the flag. Table 3 shows the characteristics of the vulnerabilities of each machine.

As for the countermeasures that can be taken in this particular environment, it is important to know that:

- The flag can be migrated to any node in normal state in the core network, but not to a machine in the honeynet.
- Through the use of appropriate flows, any node can be logically moved from the main network to the honeynet.
- Any node in the network can be isolated.

TABLE 3. Vulnerabilities and connections of the studied case. The red row represents the attacked node and the green one corresponds to the flag.

Node	BS	Expl.	Imp.	Scope	Neighbors
1	10.0	6.0	3.9	Changed	2, 3, 4, 5, 6, 7, 8
2	9.9	6.0	3.1	Changed	1, 3, 4, 5, 6, 7, 8
3	8.8	5.9	2.8	Unchanged	1, 2, 4, 5, 6, 7, 8
4	7.9	4.7	2.5	Changed	1, 2, 3, 5, 6, 7, 8
5	8.8	5.3	2.8	Changed	1, 2, 3, 4, 6, 7, 8
6	7.6	4.7	2.8	Unchanged	1, 2, 3, 4, 5, 7, 8
7	6.7	5.5	1.2	Changed	1, 2, 3, 4, 5, 6, 8
8	9.6	6.0	2.8	Changed	1, 2, 3, 4, 5, 6, 7
9	6.8	4.2	2.5	Unchanged	10, 11, 12
10	7.1	4.2	2.8	Unchanged	9, 11, 12
11	7.6	4.7	2.8	Unchanged	9, 10, 12
12	10.0	6.0	3.9	Changed	9, 10, 11

E. RESULTS

We have carried out five train-and-play sets of experiments, each one with a different random seed. In each set, the models obtained after 10^3, 10^4, and 10^5 training steps have been saved. MuZero trains both agents at the same time by self-play, so three attackers and three defenders of different levels have been obtained for each of the seeds. Then, 100 games have been played for each of the nine combination between defenders and attackers of different levels. Therefore, 900 games were played for each training seed, 4500 in total. Using the available resources, training each of the models for 10^5 training steps required more than 7 hours.

Table 4 shows the means and medians of the rewards earned by the agents and the lengths of the episodes won by each agent, for each batch of 100 games. The winning percentage (WP) of both players is also displayed. Each batch has been started with the same seed.

Prior to analyze the development of the agents, it is noteworthy that it seems to be a bias in the design of the game towards the defender, since in those games where the 10^3 training steps attacker faces a defender with that same amount of training steps, the defender usually wins. That imbalance starts to disappear as the agents are trained: after 10^4 training steps, the difference between the winning ratios of the players tends to decrease. Finally, attacker wins most of the games played between the agents trained with 10^5 steps.

Regarding the attacker, by comparing the results of the games played between the defender trained with 10^3 steps and the different attackers (first, fourth and seventh rows of the results of each seed) we can conclude that, in general, the points obtained by the attacker increase and the episode length decreases as the attacker gets additional training steps. In other words, the attacker is not only able to compromise as many hosts as possible, but it also attacks the flag following a more precise strategy. Seed A shows a small increase in the number of games won by the defender when comparing the performance of an attacker trained with $10^4 - 10^5$ steps against the least trained defender, but more tests should be performed in order to ensure that the difference is statistically significant.

TABLE 4. Results of 4500 games between different attackers (A) and defenders (D) obtained from five independent trainings.

			Reward				Episode length					
	Training steps		Mean		Median		Mean		Median		WP (%)	
	A	D	A	D	A	D	A	D	A	D	A	D
Seed A	10^3	10^3	-	0.00	-	0.0	-	119.00	-	119.0	0	1
	10^3	10^4	-	20.28	-	0.0	-	84.56	-	88.0	0	18
	10^3	10^5	-	127.44	-	133.5	-	25.08	-	17.0	0	100
	10^4	10^3	72.78	0.00	64.5	0.0	63.44	113.00	46.0	113.0	18	1
	10^4	10^4	67.64	111.20	40.0	119.0	50.36	15.40	20.0	11.0	11	5
	10^4	10^5	54.33	110.40	47.5	131.5	43.00	28.97	35.0	17.0	6	72
	10^5	10^3	123.00	0.00	135.0	0.0	51.27	72.00	52.0	56.0	91	4
	10^5	10^4	129.49	30.00	135.0	0.0	57.44	52.60	52.0	49.0	89	5
	10^5	10^5	125.94	116.00	138.0	141.5	50.96	14.00	48.0	6.0	71	26
Seed B	10^3	10^3	-	34.26	-	0.0	-	69.71	-	69.0	0	82
	10^3	10^4	-	73.84	-	81.0	-	41.76	-	33.0	0	97
	10^3	10^5	-	100.60	-	115.0	-	30.76	-	25.0	0	100
	10^4	10^3	50.88	54.39	34.0	48.0	76.38	51.63	69.0	41.0	16	54
	10^4	10^4	60.42	90.47	53.0	104.0	75.00	28.79	83.0	23.0	12	77
	10^4	10^5	43.25	109.79	26.5	127.0	46.83	24.90	32.0	11.0	12	77
	10^5	10^3	100.74	49.00	95.0	0.0	41.23	58.60	36.0	49.0	91	5
	10^5	10^4	97.70	73.00	95.0	65.0	39.73	31.27	34.0	23.0	83	15
	10^5	10^5	101.54	113.24	107.0	137.0	41.38	12.52	38.0	7.0	71	21
Seed C	10^3	10^3	-	31.38	-	0.0	-	73.32	-	70.0	0	68
	10^3	10^4	-	65.55	-	80.0	-	49.37	-	33.0	0	87
	10^3	10^5	-	98.15	-	110.5	-	29.06	-	22.0	0	100
	10^4	10^3	25.00	39.54	25.0	17.0	84.00	64.30	84.0	59.0	1	57
	10^4	10^4	-	64.09	-	66.5	-	52.09	-	43.0	0	88
	10^4	10^5	-	101.51	-	117.5	-	26.24	-	17.0	0	100
	10^5	10^3	101.83	64.33	98.0	85.0	34.02	41.67	34.0	27.0	91	9
	10^5	10^4	110.02	95.33	110.0	134.0	37.44	20.78	34.0	5.0	90	9
	10^5	10^5	118.17	118.59	123.0	134.0	42.36	11.00	38.0	5.0	83	17
Seed D	10^3	10^3	-	35.45	-	15.0	-	67.72	-	59.0	0	69
	10^3	10^4	-	130.64	-	134.0	-	27.40	-	21.0	0	100
	10^3	10^5	-	131.12	-	133.0	-	24.92	-	20.0	0	100
	10^4	10^3	85.50	55.50	83.0	49.0	67.95	55.85	62.0	41.0	42	26
	10^4	10^4	60.81	106.92	25.0	136.0	34.13	26.90	21.0	13.0	32	63
	10^4	10^5	42.67	123.71	25.0	140.0	24.00	19.00	16.0	9.0	30	62
	10^5	10^3	114.88	58.50	113.5	50.5	47.33	36.50	47.0	28.0	90	4
	10^5	10^4	114.49	87.08	120.0	140.0	47.23	27.00	42.0	9.0	75	24
	10^5	10^5	115.64	123.44	120.0	141.0	39.09	11.80	40.0	7.0	75	25
Seed E	10^3	10^3	-	-	-	-	-	-	-	-	0	0
	10^3	10^4	-	131.24	-	136.0	-	26.43	-	17.0	0	98
	10^3	10^5	-	131.34	-	136.0	-	26.32		17.0	0	100
	10^4	10^3	-	-	-	-	-	-	-	-	0	0
	10^4	10^4	-	131.24	-	136.0	-	26.43	-	17.0	0	98
	10^4	10^5	-	131.34	-	136.0	-	26.32	-	17.0	0	100
	10^5	10^3	132.68	-	151.0	-	47.90	-	48.0	-	97	0
	10^5	10^4	131.78	128.89	135.0	142.0	49.21	11.21	50.0	5.0	79	19
	10^5	10^5	127.94	113.84	151.0	104.0	46.06	15.13	46.0	11.0	69	31

Similar results are observed when we compare the performance of different attackers intruding a network defended by an agent trained with 10^4 steps: longer trainings usually lead to bigger rewards for the attacker and shorter episodes. In some cases, the attacker with 10^4 training steps seems to perform better than the one with more steps. As for the attacker's goals, there is a trade-off between reaching the flag as soon as possible (lower reward but fewer moves) and compromising as many computers as possible (higher reward but more moves and a high possibility of not winning the game thus obtaining a negative reward). That said, these differences could be the consequence of the attacker preferring to obtain a lower reward in order to ensure winning the game.

The improvement of the attacker with additional training steps can also be observed by comparing the results it obtained against the defender trained with 10^5 steps: while the less trained intruder is completely unable to reach the flag, the more trained one (with 10^5 training steps) wins most of the games in a few moves. The same could be said for improving the efficiency of the "intelligent" defender, which is the main objective of this work. It can be clearly seen that, when facing an attacker of a certain level against different defenders, those defenders with longer training achieve better performance figures: better scores, shorter games and a higher percentage of games won in all analyzed cases.

Finally, we would like to highlight that, although the quantitative results shown in Table 4 depend on the seed, the evolution shown by the agents is similar in all cases and the results of the final games—played between agents trained with 10^5 steps—are remarkably consistent.

V. CONCLUSION AND FUTURE WORK

In this paper we have presented an implementation of an intelligent system trained by MuZero that is able to mitigate intrusions in SDN networks in an autonomous way. The interaction between the attacker and the defender has been posed as a partially observed Markov game and the countermeasures have been implemented in an OpenFlow-based SDN using the Ryu controller.

The results presented in Section IV indicate that a defender sufficiently trained may be successful at choosing an appropriate intrusion mitigation strategy. Similarly, a well trained attacker my be successful evading the countermeasures of the defender. The case study presented here is based on a relatively small network, but it can be easily adapted to work with different network topologies. We consider that the obtained results are highly satisfactory, taking into account that we have posed this problem as a *stochastic* game and neither the MuZero algorithm itself nor the used implementation have been designed to train agents in this kind of games. We believe that better results could be obtained with a modified version of MuZero designed to play non-deterministic games, which we leave as a possible line of future research.

A defender trained using this game could be integrated with intrusion detection and host state monitoring systems.

TABLE 5. Hyperparameters selection used in all trainings.

Hyperparameter	Value
Discount	0.997
TD steps	150
Replay buffer size	$1 \cdot 10^5$
Batch size	250
Unroll steps	5
MuZero Reanalyze	Enabled
Network type	fullyconnected
Encoding size	256
# of representation layers	32
# of dynamics layers	32
# of reward layers	16
# of value layers	16
# of policy layers	16
Optimizer	Adam [57]
Initial learning rate	$4 \cdot 10^{-4}$
Decay rate	0
L2 reg. weight	$1 \cdot 10^{-4}$
Simulations	75
Dirichlet α	0.25
Exploration factor	0.3
pUCT c_1	1.25
pUCT c_2	19652
Temperature	1.0

Thus, once an attack is detected, the trained model could be used to autonomously apply the countermeasures required to mitigate the intrusion. A possible application of this implementation is the response to zero-day attacks.

Moreover, the work presented in this article can easily be applied to other network configurations. For instance, if the network presents a sub network which has a high probability of being attacked—maybe because it is accessible for external users—, the option of migrating the flag to one of those servers should be removed. In conclusion, the game is able to represent several scenarios with minor modifications.

Nevertheless, larger and more complex networks, or an increase in the number of available actions, would generate game environments that agents would find difficult to deal with unless models are trained over a much larger number of steps. These complex environments would require extensive tuning of MuZero and an increase in the computational resources required. A scalability study should be performed in a high-performance computing (HCP) platform.

APPENDIX A
HYPERPARAMETERS

See Table 5.

REFERENCES

[1] S. Morgan, "2021 report: Cyberwarfare in the C-suite," Cybersecur. Ventures, Long Island, NY, USA, 2021. [Online]. Available: https://cybersecurityventures.com/wp-content/uploads/2021/01/Cyberwarfare-2021-Report.pdf

[2] S. Mahdavifar and A. A. Ghorbani, "Application of deep learning to cybersecurity: A survey," *Neurocomputing*, vol. 347, pp. 149–176, Jun. 2019.

[3] N. Sultana, N. Chilamkurti, W. Peng, and R. Alhadad, "Survey on SDN based network intrusion detection system using machine learning approaches," *Peer Peer Netw. Appl.*, vol. 12, no. 2, pp. 493–501, Jan. 2019.

[4] M. Sainz, I. Garitano, M. Iturbe, and M. Zurutuza, "Deep packet inspection for intelligent intrusion detection in software-defined industrial networks: A proof of concept," *Logic J. IGPL*, vol. 28, no. 4, pp. 461–472, Dec. 2020.

[5] A. S. Da Silva, J. A. Wickboldt, L. Z. Granville, and A. Schaeffer-Filho, "ATLANTIC: A framework for anomaly traffic detection, classification, and mitigation in SDN," in *Proc. IEEE/IFIP Netw. Oper. Manage. Symp. (NOMS)*, Apr. 2016, pp. 27–35.

[6] C. J. Chung, P. Khatkar, T. Xing, J. Lee, and D. Huang, "NICE: Network intrusion detection and countermeasure selection in virtual network systems," *IEEE Trans. Dependable Secure Comput.*, vol. 10, no. 4, pp. 198–211, Jul. 2013.

[7] K. Hammar and R. Stadler, "Finding effective security strategies through reinforcement learning and self-play," in *Proc. 16th Int. Conf. Netw. Service Manage. (CNSM)*, Nov. 2020, pp. 1–9.

[8] J. Schrittwieser, I Antonoglou, T Hubert, K Simonyan, L Sifre, S Schmitt, A Guez, E Lockhart, D Hassabis, T Graepel, and T. Lillicrap, "Mastering Atari, Go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, pp. 604–609, 2020.

[9] Q. Duan and M. Toy, *Software-Defined Networking*. Norwood, MA, USA: Artech House, 2016.

[10] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 3, pp. 1617–1634, 3rd Quart., 2014.

[11] C. Trois, M. D. Del Fabro, L. C. E. de Bona, and M. Martinello, "A survey on SDN programming languages: Toward a taxonomy," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 4, pp. 2687–2712, 4th Quart., 2016.

[12] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Apr. 2008.

[13] *OpenFlow Switch Specification*, Open Netw. Found., Menlo Park, CA, USA, Oct. 2013.

[14] S. Ahmad and A. H. Mir, "Scalability, consistency, reliability and security in SDN controllers: A survey of diverse SDN controllers," *J. Netw. Syst. Manage.*, vol. 29, no. 1, pp. 1–59, Jan. 2021.

[15] L. Zhu, M. M. Karim, K. Sharif, F. Li, X. Du, and M. Guizani, "SDN controllers: Benchmarking & performance evaluation," 2019, *arXiv:1902.04491*. [Online]. Available: https://arxiv.org/abs/1902.04491

[16] N. Z. Bawany, J. A. Shamsi, and K. Salah, "DDoS attack detection and mitigation using SDN: Methods, practices, and solutions," *Arabian J. Sci. Eng.*, vol. 42, no. 2, pp. 425–441, 2017.

[17] T. Xing, D. Huang, L. Xu, C.-J. Chung, and P. Khatkar, "SnortFlow: A OpenFlow-based intrusion prevention system in cloud environment," in *Proc. 2nd GENI Res. Educ. Exp. Workshop*, Mar. 2013, pp. 89–92.

[18] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *J. Artif. Intell. Res.*, vol. 4, no. 1, pp. 237–285, Jan. 1996.

[19] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Process. Mag.*, vol. 34, no. 6, pp. 26–38, Nov. 2017.

[20] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[21] R. Bellman, "A Markovian decision process," *Indiana Univ. Math. J.*, vol. 6, no. 4, pp. 679–684, Apr. 1957.

[22] R. Bellman, "Dynamic programming," *Science*, vol. 153, nos. 37–31, pp. 34–37, 1966.

[23] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.

[24] M. Bowling and M. Veloso, "An analysis of stochastic game theory for multiagent reinforcement learning," School Comput. Sci., Carnegie-Mellon Univ., Pittsburgh, PA, USA Tech. Rep. CMU-CS-00-165, 2000.

[25] A. Nowé, P. Vrancx, and Y.-M. D. Hauwere, *Game Theory and Multi-Agent Reinforcement Learning*. Berlin, Germany: Springer, 2012.

[26] C. B. Browne E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of Monte Carlo tree search methods," *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. 1, pp. 1–43, Mar. 2012.

[27] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Machine Learning Proceedings*. Amsterdam, The Netherlands: Elsevier, 1994, pp. 157–163.

[28] B. Molina-Coronado, U. Mori, A. Mendiburu, and J. Miguel-Alonso, "Survey of network intrusion detection methods from the perspective of the knowledge discovery in databases process," *IEEE Trans. Netw. Service Manage.*, vol. 17, no. 4, pp. 2451–2479, Dec. 2020.

[29] P. Hadem, D. K. Saikia, and S. Moulik, "An SDN-based intrusion detection system using SVM with selective logging for IP traceback," *Comput. Netw.*, vol. 191, May 2021, Art. no. 108015.

[30] A. Yazdinejadna, R. M. Parizi, A. Dehghantanha, and M. S. Khan, "A kangaroo-based intrusion detection system on software-defined networks," *Comput. Netw.*, vol. 184, Jan. 2021, Art. no. 107688.

[31] A. D. R. L. Ribeiro, R. Y. C. Santos, and A. C. A. Nascimento, "Anomaly detection technique for intrusion detection in SDN environment using continuous data stream machine learning algorithms," in *Proc. IEEE Int. Syst. Conf. (SysCon)*, Apr. 2021, pp. 1–7.

[32] M. Ajdani and H. Ghaffary, "Design network intrusion detection system using support vector machine," *Int. J. Commun. Syst.*, vol. 34, no. 3, Feb. 2021, Art. no. e4689.

[33] O. Rahman, M. A. G. Quraishi, and C.-H. Lung, "DDoS attacks detection and mitigation in SDN using machine learning," in *Proc. IEEE World Congr. Services (SERVICES)*, vol. 2642, Jul. 2019, pp. 184–189.

[34] *CVE—Common Vulnerabilities and Exposures*, Mitre Corp., McLean, VA, USA, Dec. 2020.

[35] E. Ukwandu, M. A. B. Farah, H. Hindy, D. Brosset, D. Kavallieros, R. Atkinson, C. Tachtatzis, M. Bures, I. Andonovic, and X. Bellekens, "A review of cyber-ranges and test-beds: Current and future trends," *Sensors*, vol. 20, no. 24, p. 7148, Dec. 2020.

[36] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*. [Online]. Available: http://arxiv.org/abs/1707.06347

[37] R. J. Williams, *Reinforcement-Learning Connectionist Systems*. Toronto, ON, Canada: College of Computer Science, Northeastern Univ., 1987.

[38] M. Campbell, A. J. Hoane, Jr., and F.-H. Hsu, "Deep blue," *Artif. Intell.*, vol. 134, nos. 1–2, pp. 57–83, Jan. 2002.

[39] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, and S. Dieleman, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[40] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, and A. Bolton, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[41] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, P. T. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *CoRR*, vol. abs/1712.01815, pp. 1–19, Dec. 2017.

[42] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, and S. Legg, "IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1407–1416.

[43] S. Kapturowski, G. Ostrovski, J. Quan, R. Munos, and W. Dabney, "Recurrent experience replay in distributed reinforcement learning," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–19.

[44] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. van Hasselt, and D. Silver, "Distributed prioritized experience replay," 2018, *arXiv:1803.00933*. [Online]. Available: http://arxiv.org/abs/1803.00933

[45] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proc. 9th ACM SIGCOMM Workshop Hot Topics Netw. (Hotnets)*, 2010, pp. 1–6.

[46] K. Morita, I. Yamahata, and V. Linux, "Ryu: Network operating system," in *Proc. OpenStack Design Summit Conf.*, 2012, pp. 1–7.

[47] J. Ren, C. Zhang, and Q. Hao, "A theoretical method to evaluate honeynet potency," *Future Gener. Comput. Syst.*, vol. 116, pp. 76–85, Mar. 2021.

[48] L. Spitzner, "The honeynet project: Trapping the hackers," *IEEE Security Privacy*, vol. 1, no. 2, pp. 15–23, Mar. 2003.

[49] E. Dart, L. Rotman, B. Tierney, M. Hester, and J. Zurawski, "The science DMZ: A network design pattern for data-intensive science," *Sci. Program.*, vol. 22, no. 2, pp. 173–185, 2014.

[50] H. Booth, D. Rike, and G. Witte, "The national vulnerability database (NVD): Overview," ITL Bull., Nat. Inst. Standards Technol., Gaithersburg, MD, USA, 2013.

[51] *Common Vulnerability Scoring System Version 3.1: Specification Document*, Forum of Incident Response and Security Teams, Cary, NC, USA, Jun. 2019.

[52] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA, USA: CreateSpace, 2009.

[53] A. H. W. Duvaud. (2019). *MuZero General: Open Reimplementation of MuZero*. [Online]. Available: https://github.com/werner-duvaud/muzero-general

[54] RYU Project Team. (Feb. 2014). *RYU SDN Framework*. [Online]. Available: https://book.ryu-sdn.org/en/Ryubook.pdf

[55] S. Bhatt, P. K. Manadhata, and L. Zomlot, "The operational role of security information and event management systems," *IEEE Security Privacy*, vol. 12, no. 5, pp. 35–41, Sep. 2014.

[56] A. B. Asif, M. Imran, N. Shah, M. Afzal, and H. Khurshid, "ROCA: Auto-resolving overlapping and conflicts in access control list policies for software defined networking," *Int. J. Commun. Syst.*, vol. 34, no. 9, p. e4815, Jun. 2021.

[57] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," presented at the 3rd Int. Conf. Learn. Represent., San Diego, CA, USA, 2015.

JON GABIRONDO-LÓPEZ received the degree in physics, the degree in electronical engineering, and the degree (advanced) in computational engineering and intelligent systems from the University of the Basque Country (UPV/EHU), in 2020 and 2021, respectively.

During the 2018–2019 academic year, he obtained an IKASIKER Fellowship to work at the Physics Department, UPV/EHU, where he carried out a research project aimed at the computational study of optical properties in metals. During the 2020–2021 academic year, he was a Researcher on reinforcement learning and cybersecurity with the Department of Digital Security, Vicomtech. He is currently part of the Physics Department, UPV/EHU. He develops web applications for the Bilbao Crystallographic Server.

JON EGAÑA received the Telecommunication Engineering degree and the advanced degree in telecommunication engineering from UPV/EHU, in 2014 and 2016, respectively.

He is currently with the Department of Digital Security, Vicomtech, where he works as a Researcher in the field of data analytics for cybersecurity. He is an Active Member of the 5GPPP Security Work Group.

JOSE MIGUEL-ALONSO (Member, IEEE) graduated in computer science from the University of the Basque Country (UPV/EHU), Spain, in 1989, and received the Ph.D. degree from UPV/EHU, in 1996.

He is currently a Full Professor with the Department of Computer Architecture and Technology, UPV/EHU. He is a member of the Intelligent Systems Group, UPV/EHU. He carries out research related to networks and parallel-distributed systems, in areas such as cybersecurity, performance modeling (with focus on the interconnection network), resource management in supercomputers, cloud infrastructures and high-performance scientific, and technical applications. He has published 2 books, 33 journal articles, and 30 papers in international conferences. He is a member of the IEEE Computer Society and the HiPEAC Network of Excellence on High Performance and Embedded Architecture and Compilation.

RAUL ORDUNA URRUTIA received the degree in computer engineering from the University of the Basque Country (UPV/EHU), in 1999, and the Ph.D. degree from the Public University of Navarre (UPNA), in 2010.

From 2001 to 2018, he has worked in private companies as S21se, Panda Security, or Tracasa in cybersecurity and innovation positions. He is currently the Digital Security Director at Vicomtech. He has taken part or led projects related with ethical hacking, forensic analysis, malware analysis, access control, and cryptography. The current research lines are focused on anomaly detection in information systems and communication networks, automatic responses, identity management using biometrics and federated systems, and secure traceability systems.