Konputazio Ingeniaritza eta
Sistema Adimentsuak Unibertsitate Masterra

Máster Universitario en Ingeniería Computacional
y Sistemas Inteligentes

Konputazio Zientziak eta Adimen Artifiziala Saila

Departamento de Ciencias de la Computación e Inteligencia Artificial

Master Tesia
Tesis de Máster

Mitigación autónoma de intrusiones en redes SDN utilizando agentes inteligentes basados en MuZero

Jon Gabirondo López

Zuzendaritza
Dirección

Jose Miguel Alonso
Konputagailuen Arkitektura eta Teknologia Saila
Euskal Herriko Unibertsitatea (UPV/EHU)
Departamento de Arquitectura y Tecnología de Computadores
Universidad del País Vasco (UPV/EHU)

Jon Egaña Zubia
Segurtasun Digitala Saila
Vicomtech Fundazioa
Departamento de Seguridad Digital
Fundación Vicomtech

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA
Konputazio Ingeniaritza eta Sistema Adimentsuak Unibertsitate Masterra
Máster Universitario en Ingeniería Computacional y Sistemas Inteligentes

Resumen

El paradigma de las redes definidas por software (SDN) permite desarrollar sistemas que monitorizan y controlan el tráfico de una red de forma centralizada, facilitando la implementación de sistemas basados en el aprendizaje automático que detectan y mitigan intrusiones. Este trabajo presenta un sistema inteligente capaz de decidir qué contramedidas tomar para mitigar una intrusión en una red definida por software. La interacción entre el intruso y el defensor se plantea como un juego de Markov parcialmente observado y el modelo se entrena mediante el algoritmo de aprendizaje por refuerzo MuZero. Una vez entrenado, el modelo se integra con un controlador SDN para así implementar las contramedidas elegidas por el defensor en la red real. A la hora de evaluar el rendimiento del modelo, se han enfrentado entre ellos a atacantes y defensores con pasos de entrenamiento diferentes y se han registrado las puntuaciones obtenidas por cada uno de ellos, la duración de las partidas y el ratio de partidas ganadas por cada agente. Los resultados muestran que un defensor suficientemente entrenado es capaz de decidir las contramedidas que minimizan el impacto de la intrusión, aislando al atacante y evitando que comprometa máquinas clave de la red.

Palabras clave: Agentes inteligentes, aprendizaje por refuerzo, ciberseguridad, juegos de Markov, MuZero, OpenFlow, redes definidas por software, respuesta autómática.

Índice general

Capítulo 1

Introducción

La cantidad de usuarios de Internet ha crecido considerablemente durante los últimos años y cada vez son más los servicios—desde comercios electrónicos a operaciones bancarias—proporcionados a través de la red. En consecuencia, no solo la cantidad, si no también la gravedad de los ciberataques sufridos por organizaciones y empresas ha ido en aumento, causando pérdidas millonarias [1].

Tradicionalmente, los sistemas de seguridad de las empresas se han diseñado e implementado de forma manual por personal experto y la respuesta a ataques o intrusiones también se ha llevado a cabo por esos técnicos. El nuevo paradigma de infraestructuras de red presentado por las redes definidas por software (SDN) ha permitido desarrollar sistemas de detección y mitigación de ataques basados en técnicas de aprendizaje automático [2, 3, 4, 5, 6]. Comparados con los sistemas de detección convencionales o con las estrategias de mitigación dirigidas por los humanos, estos sistemas no solo permiten detectar de una manera más rápida y precisa los ataques, si no que implementan las contramedidas de forma autónoma y automática, minimizando el tiempo de reacción y de respuesta ante un ataque, reduciendo así el daño causado en la red.

Por otro lado, el desarrollo de algoritmos de aprendizaje por refuerzo capaces de superar marcas humanas en juegos de mesa—como el ajedrez o el Go—, ha alentado la idea de plantear la intrusión en una red como un juego de Markov y de utilizar esos algoritmos para entrenar agentes inteligentes que pueden buscar de forma autónoma estrategias de seguridad para mitigar intrusiones [7].

En este trabajo, desarrollado en el Departamento de Seguridad Digital de Vicomtech y situado dentro del proyecto Égida[1] impulsado por el Ministerio de Ciencia e Innovación y el Centro de Desarrollo Tecnológico Industrial (CDTI), se presenta un sistema inteligente capaz de elegir y ejecutar las contramedidas adecuadas para mitigar una intrusión en una red SDN, minimizando el daño causado por el ataque. El juego se ha diseñado teniendo en cuenta que debe representar el estado de los nodos de una red real y que las acciones del defensor deben poder implementarse mediante un controlador SDN. El modelo se ha entrenado utilizando el algoritmo *model-based* de aprendizaje por refuerzo MuZero [8].

Las contribuciones presentadas en este trabajo son las siguientes:

- El diseño de un juego de Markov capaz de representar las vulnerabilidades de una red de ordenadores.
- La implementación de las contramedidas necesarias mediante un controlador SDN.
- La implementación de un entorno virtual en el que se llevan a cabo de manera autónoma las contramedidas seleccionadas por el defensor en una red SDN emulada.

El resto de este informe está estructurado de la siguiente manera: la Sección 2 recoge los conceptos básicos relacionados con las redes SDN y con el aprendizaje por refuerzo, profundizando en el funcionamiento del algoritmo MuZero. También muestra los últimos trabajos relacionados con ambos campos. La Sección 3 presenta la propuesta realizada en este trabajo, explicando el funcionamiento del juego y su integración con una red SDN. En la Sección 4 se muestra la información referente al entrenamiento del modelo y los resultados obtenidos. Finalmente, en la Sección 5 se evalúan dichos resultados y se enumeran las posibles mejoras que podrían llevarse a cabo en el futuro.

[1] https://egidacybersecurity.com/

Capítulo 2

Contexto teórico y estado del arte

2.1. Contexto teórico

2.1.1. Redes definidas por software

Durante los últimos años, las aplicaciones y los servicios puestos a disposición del usuario a través de Internet se han vuelto cada vez más complejos y exigentes. Esto ha dejado en evidencia la necesidad de un cambio de paradigma en el mundo de las infraestructuras de red, puesto que las redes convencionales no disponen del dinamismo y la adaptabilidad que requieren las nuevas plataformas [9].

Las redes convencionales están compuestas por elementos (como *routers* o *switches*) habitualmente tratados como *cajas negras* que se basan en interfaces de control limitados o específicos del fabricante. Esos dispositivos suelen configurarse de manera individual y las decisiones sobre la gestión del tráfico se toman directamente en cada dispositivo, fusionando el *plano de control* (*control plane*) encargado de tomar dichas decisiones con el *plano de datos* (*data plane*) compuesto por los elementos de la red [10, 11]. La falta de independencia entre ambos planos dificulta enormemente la adaptación dinámica de la red para satisfacer las necesidades de las aplicaciones o para hacer frente a determinados eventos. Este problema es una de las causas principales de la "osificación de Internet": el desarrollo de nuevos protocolos e infraestructuras se ha visto gravemente entorpecido por la propia arquitectura de los elementos de la red [10, 12].

Las redes definidas por software o SDN (de sus siglas en inglés de *Software-Defined Networking*) proponen sistemas en los que el plano de control y el de datos aparecen completamente desacoplados, permitiendo la configuración centralizada de las infraestructuras, tal y como muestra la Figura 2.1. Aunque existen diferentes arquitecturas de SDN, en este trabajo se contemplan únicamente las que utilizan el protocolo OpenFlow [12] en la comunicación entre el plano de control y el de datos, pues es el protocolo más utilizado y ha sido respaldado por la ONF[1] (*Open Networking Foundation*), la organización sin ánimo de lucro dedicada al desarrollo y estandarización de las redes SDN [11]. En dichas arquitecturas, los elementos que forman la red (conocidos como OpenFlow switches) pasan a encargarse únicamente del reenvío del tráfico, mientras que las decisiones son tomadas por el controlador de la red. Para ello, el controlador instala *tablas de flujos* en los switches mediante el protocolo OpenFlow. Las tablas están formadas por flujos que determinan cómo se procesan y reenvían los paquetes que cumplen ciertos criterios.

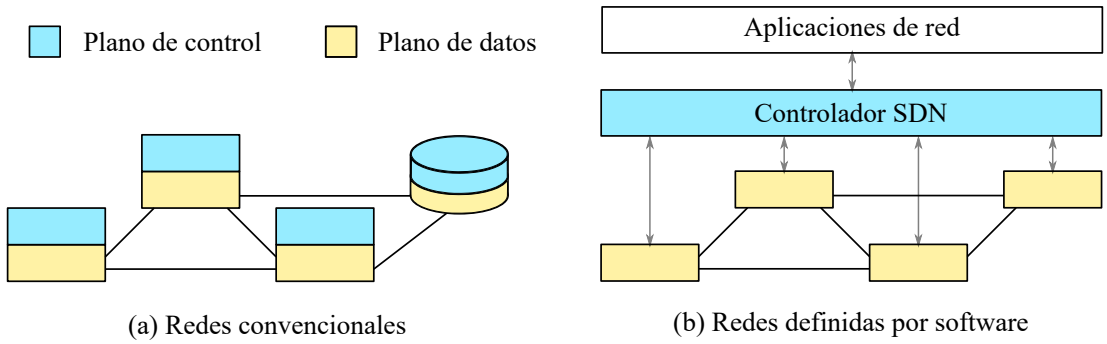


Figura 2.1: Diferencias estructurales entre una red convencional y una red definida por software.

Cada uno de esos flujos está compuesto por una serie de campos, tal y como se muestra en la Tabla 2.1. Los *Match Fields* son las condiciones que tiene que cumplir el paquete entrante (como la dirección IP de origen o el puerto de entrada, por ejemplo) para que se le empareje con las instrucciones definidas en el campo *Instructions*. Esas instrucciones pueden incluir el bloqueo de todo el tráfico proveniente de algún puerto o el reenvío de los paquetes al controlador, por ejemplo. En el caso de que el paquete cumpla los requisitos de más de un flujo, se

[1] http://www.opennetworking.org/

ejecutará el que tenga una prioridad mayor de acuerdo con el campo *Priority*. El campo *Counters* recoge el número de paquetes procesados por ese flujo, *Timeouts* define cuánto tiempo debe pasar sin que ningún paquete se empareje con el flujo antes de eliminarlo y *Cookie* es, simplemente, un identificador impuesto por el controlador [13].

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie

Tabla 2.1: Componentes de un flujo de OpenFlow.

Si un paquete recibido no coincide con ningún requisito y se queda sin emparejar, se ejecutan las instrucciones definidas en el flujo *table-miss*, que puede incluir acciones como eliminar el paquete o dejar que el controlador decida qué hacer [10, 14].

La Figura 2.2 muestra la estructura que presentan este tipo de redes SDN, compuesta por tres componentes principales y dos interfaces que permiten la comunicación entre ellos[2].

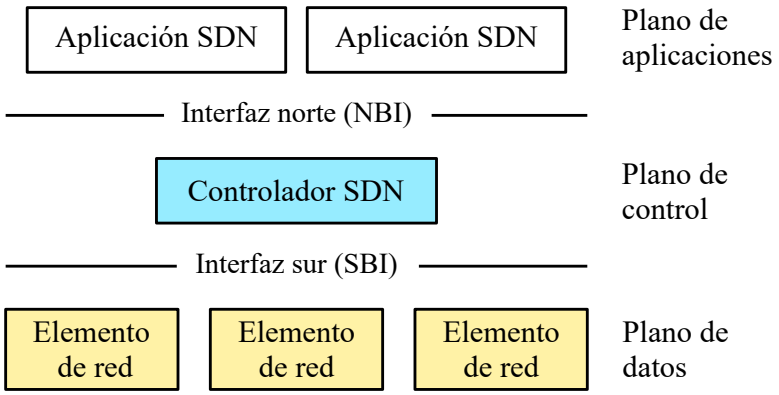


Figura 2.2: Diagrama de los componentes de una red SDN y de las interfaces entre ellos.

El controlador utiliza la interfaz sur (SBI, de sus siglas en inglés de *Southbound Interface*) para comunicarse con los dispositivos de la red. De esta manera, el controlador gestiona la estructura de la red instalando flujos en los switches y recoge

[2] Existen estructuras más complejas que incluyen varios controladores y más interfaces, pero en esta explicación se ha considerado únicamente el caso de un solo controlador.

información sobre el tráfico del plano de datos. La interfaz no depende del proveedor del equipo y el protocolo más utilizado es el ya mencionado OpenFlow [15].

Similarmente, el controlador se comunica con el *plano de aplicaciones* (*application plane*) mediante la interfaz norte (NBI, de sus siglas en inglés de *Northbound Interface*). Dicho plano está formado por diversas aplicaciones SDN que implementan estrategias de control y gestión del tráfico como balanceos de cargas o cortafuegos. Por lo tanto, la interfaz norte permite a las aplicaciones procesar la información del plano de datos que recibe a través del controlador, realizar acciones a un alto nivel y hacer que el controlador las ejecute en la infraestructura. La comunicación entre las aplicaciones y el plano de control depende del controlador, pero muchas veces se utilizan APIs REST [16].

Es interesante presentar un flujo de trabajo entero para subrayar las acciones realizadas por cada componente y la comunicación entre ellos. Supongamos que la comunicación en la interfaz norte se realiza mediante una API REST y que la interfaz sur se basa en el protocolo OpenFlow. Digamos que una de las aplicaciones SDN, tras analizar los datos recibidos, decide que la máquina con dirección IP `192.168.2.56`, conectada a los switches del plano de datos, debería ser aislada del resto. Entonces, la aplicación mandaría al controlador un mensaje como `aislar 192.168.2.56` y este traduciría dicha acción al protocolo OpenFlow, resultando en la instalación en los switches de flujos parecidos a los siguientes:

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie
IP src: `192.168.2.56`	10	0	`DROP`	1000	1
IP dst: `192.168.2.56`	10	0	`DROP`	1000	2

Tabla 2.2: Ejemplo de flujos que bloquearían todos los paquetes entrantes y salientes de la máquina con la dirección IP 192.168.2.56.

Una propiedad importante de las SDN es su capacidad de reaccionar ante diferentes eventos. El controlador puede realizar una configuración inicial de la red de forma proactiva. Sin embargo, esa configuración puede modificarse dinámicamente (añadiendo o quitando flujos) en función del tráfico de la red. Por esta razón, las redes SDN se han utilizado en múltiples ocasiones para diseñar sistemas capaces de detectar y responder dinámicamente a ciberataques [2, 3, 4, 5, 6, 17, 18].

En resumen, el paradigma planteado por las redes SDN permite desplegar infraestructuras basadas en dispositivos que no dependen de las interfaces impuestas por los fabricantes, estableciendo una entidad que puede monitorizar e implementar dinámicas de control de una forma centralizada.

2.1.2. Aprendizaje por refuerzo y juegos de Markov

El aprendizaje por refuerzo (RL, de sus siglas en inglés de *Reinforcement Learning*) es una rama del aprendizaje automático que estudia cómo un agente aprende a tomar decisiones siguiendo una estrategia de ensayo y error [19].

La Figura 2.3, basada en una figura del libro [20], muestra los elementos principales de los modelos de RL. El *agente* es el sujeto del entrenamiento y el que tiene que aprender a tomar decisiones. El *entorno*, en cambio, representa el mundo con el que interactúa el agente [20]. En cada interacción t, el agente recibe una *observación*—total o parcial—$o_t \in \mathcal{O}$ del estado del entorno $s_t \in \mathcal{S}$, donde $\mathcal{O}$ es el conjunto de observaciones del conjunto de estados posibles $\mathcal{S}$. En base a la observación decide qué *acción* $a_t \in \mathcal{A}(s_t)$ tomar, donde $\mathcal{A}(s_t)$ es el conjunto de acciones posibles en el estado s_t. El entorno puede cambiar ante la acción del agente o de manera independiente. Una vez ejecutada la acción, el agente recibe una recompensa numérica r_{t+1} y una nueva observación del entorno, permitiéndole evaluar la acción realizada y dar un nuevo paso.

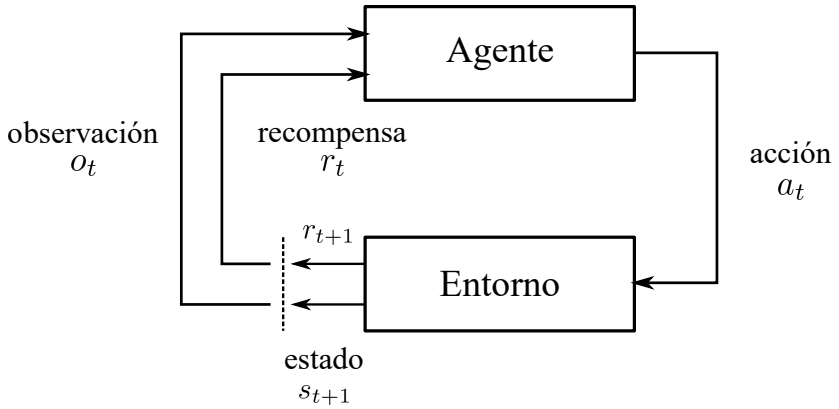


Figura 2.3: Resumen de los elementos principales de los problemas de aprendizaje por refuerzo y la interacción entre ellos.

Merece la pena remarcar la diferencia entre el *estado* y la *observación* del entorno. El estado representa toda la información del entorno y, en consecuencia, lo define totalmente. Una observación, en cambio, es una vista parcial que contiene, generalmente, solo una parte de la información referente al estado. Si la observación incluye toda la información del estado, el entorno se denomina como *completamente observado*. Si por el contrario la información es parcial, es un entorno *parcialmente observado*. En muchas ocasiones suele utilizarse directamente el término *estado* para referirse a la *observación*, y se utiliza el símbolo s_t en vez de o_t. En este caso, se ha optado por una expresión explícita de la observación por dos razones principales: la primera es que una de las claves del modelo implementado en la Sección 3.2 es que se basa en un entorno parcialmente observado. La segunda es que los autores del algoritmo utilizado para entrenar el modelo, el algoritmo MuZero de DeepMind, utilizan dicha notación en su artículo [8].

La *política* (*policy*, en inglés) es la estrategia que determina qué acción debería tomar el agente dada una observación. Esa función se actualiza según va entrenándose el modelo. En general, la política π es la probabilidad de que el agente tome una acción a_t basándose en la observación o_t. La política π suele depender de una serie de parámetros θ, por lo que la representación completa sería $\pi_\theta(a_t \mid o_t)$.

La sucesión de estados visitados y acciones tomadas $\tau = (s_0, a_0, s_1, a_1, ...)$ suele denominarse como *trayectorias* o *episodios* y el objetivo principal del agente es maximizar la suma de todas las recompensas obtenidas en un episodio. A esta suma de recompensas se le denomina como *retorno*. En muchas ocasiones, sobre todo en entornos en los que no hay una etapa final—en controladores eléctricos, por ejemplo—se utiliza una estrategia llamada *discounted return*, que hace que las recompensas obtenidas varios pasos atrás valgan menos que las cercanas en el tiempo. Dicho retorno se define de la siguiente manera

$$R(\tau) = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.1}$$

donde γ es un parámetro, $0 \leq \gamma \leq 1$, conocido como *velocidad de descuento* o *discount rate*.

Más allá de la intuición establecida hasta hora, esos procesos estudiados en RL son Procesos de Decisión de Markov (MDP, de sus siglas en inglés de *Markov Decision Process*) que se definen mediante una tupla de seis componentes como la mostrada en la Ecuación 2.2 [21].

$$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}_{ss'}, \mathcal{R}^a_{ss'}, \gamma, \rho_0 \rangle \tag{2.2}$$

siendo

$$\mathcal{P}^a_{ss'} = \mathbb{P}\left[s_{t+1} = s' \mid s_t = s, a_t = a\right] \tag{2.3}$$

$$\mathcal{R}^a_{ss'} = \mathbb{E}\left[r_{t+1} \mid a_t = a, s_t = s, s_{t+1} = s'\right] \tag{2.4}$$

donde $\mathcal{P}^a_{ss'}$ es la probabilidad de llegar a un estado s' al aplicar una acción a en el estado s y $\mathcal{R}^a_{ss'}$ es la recompensa esperada tras esa transición. El término $\rho_0 : \mathcal{S} \mapsto [0, 1]$ es la distribución del estado inicial [7]. El origen del nombre de estos procesos es que la trasición del estado s_t al estado s_{t+1} cumple la propiedad de Markov (Ecuación 2.5): solo depende de los estados mencionados y no del resto de estados anteriores [20].

$$\mathbb{P}\left[s_{t+1} \mid s_t\right] = \mathbb{P}\left[s_{t+1} \mid s_1, ..., s_t\right] \tag{2.5}$$

Como ya se ha mencionado en la introducción, el modelo desarrollado es un juego de Markov basado en un Proceso de Decisión Parcialmente Observado (POMDP, de sus siglas en inglés de *Partially Observed Markov Decision Process*), en el que el agente toma las decisiones en base a la observación que tiene del sistema y no directamente en el estado. En un momento t, la observación $o_t \in \mathcal{O}$ se relaciona con el estado $s_t \in \mathcal{S}$ mediante la función $\mathcal{Z}$, por lo que un POMDP se define de la siguiente manera:

$$\mathcal{M}_{\mathcal{P}} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}_{ss'}, \mathcal{R}^a_{ss'}, \gamma, \rho_0, \mathcal{O}, \mathcal{Z} \rangle \tag{2.6}$$

donde $\mathcal{O}$ es el conjunto de observaciones del sistema y $\mathcal{Z} = \mathbb{P}\left[o \mid s\right]$ es la probabilidad de observar o en el estado s.

Uniendo ambos campos, el problema planteado en el aprendizaje por refuerzo puede entenderse como la búsqueda de la política óptima π^* que maximiza el valor esperado de la suma de recompensas de un MDP para una cantidad máxima de pasos T:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r_{t+1}\right] \tag{2.7}$$

En los algoritmos de aprendizaje reforzado se definen funciones evaluadoras que calculan el retorno esperado de las acciones llevadas a cabo desde un estado inicial y siguiendo cierta política. Formalmente, en el caso de MDPs, la función $V_\pi(s)$ llamada *state-value function for policy* π se define como

$$V_\pi(s) = \mathbb{E}_\pi\left[R \mid s_t = s\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right] \tag{2.8}$$

y representa el retorno esperado cuando el agente sigue una política π partiendo de un estado s.

Análogamente, también se define el valor Q de tomar una acción a en un estado s como

$$Q_\pi(s,a) = \mathbb{E}_\pi\left[R \mid s_t = s, a_t = a\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right] \tag{2.9}$$

que representa el retorno esperado obtenido al seguir una política π tras realizar la acción a en el estado s. Esa función se denomina como *action-value function for policy* π.

Las *ecuaciones de Bellman* expuestas en las Ecuaciones 2.10 y 2.11 muestran que el valor de un estado o de una pareja estado-acción es el retorno esperado fruto de seguir la política optima π^* definida en la Ecuación 2.7 [22].

$$V^*(s) = \max_a \ \mathbb{E}\left[r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a\right] \tag{2.10}$$

$$Q^*(s,a) = \mathbb{E}\left[r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \mid s_t = s, a_t = a\right] \tag{2.11}$$

Recordemos que el objetivo final de un algoritmo de aprendizaje reforzado es encontrar la política óptima π^*. Para MDPs finitos, la Ecuación 2.10 tiene una

única solución, por lo que una vez calculado V^* es relativamente sencillo obtener la política óptima, ya que si se utiliza dicha función para evaluar las acciones a corto plazo (evaluando el estado s al que llega tras realizar a) la política seguida al tomar la mejor opción a cada paso es la política óptima a largo plazo [20].

Similarmente, utilizando Q^*, el agente ya no tiene que buscar el nuevo estado s que maximice $V(s)$, si no que le basta con buscar la acción a que resulte en un mayor $Q(a)$. Esto permite que el agente tome decisiones sin tener que conocer todos los estados consecutivos, esto es, sin tener que conocer las dinámicas del entorno con el que interactúa.

La estimación de esos valores es una parte fundamental de los algoritmos sin modelos (*model-free*), que buscan encontrar la política óptima sin tener un modelo del entorno, tal y como hace el algoritmo *Q-learning* [23], por ejemplo. Por otro lado, los algoritmos basados en modelos (*model-based*) también utilizan dichas funciones para construir modelos que tienen los mismo valores que el entorno original y que, por lo tanto, son equivalentes. El algoritmo MuZero [8], explicado en más detalle en la Sección 2.2.2, es uno de los ejemplos más recientes de los algoritmos basados en modelos.

Finalmente, es importante mencionar que los procesos de decisión en los que participan múltiples agentes se estudian dentro de la *teoría de juegos* [24]. Estos agentes interactúan con el entorno de manera simultánea o por turnos y obtienen cada uno su recompensa correspondiente, refinando así su política. A la hora de diseñar un juego, hay que definir las características que, además, sirven para clasificarlo [25]:

- **Suma de las recompensas:** si la suma de las recompensas de todos los jugadores es 0 o no. En juegos de dos jugadores, las recompensas suman 0 si ambos están estrictamente compitiendo el uno contra el otro.
- **Información:** si el estado del juego es completamente o parcialmente observable por los jugadores.
- **Determinismo:** si el resultado del juego depende o no en alguna medida en la suerte.

- **Secuencialidad:** si los agentes interactúan de manera secuencial o simultánea.
- **Discretitud:** si las acciones se aplican en tiempo real o no.

Concretamente, los juegos de Markov son el contexto teórico de los algoritmos de aprendizaje reforzado *multiagente* [26]. Un juego de Markov con N agentes se define mediante una tupla similar a la expuesta en la Ecuación 2.5. Comparado con un MDP, el juego de Markov definido por la Ecuación 2.12 presenta una lista de conjuntos de acciones posibles a realizar por los N agentes en la que a un agente $i \in [1, N]$ le corresponde su conjunto $\mathcal{A}_i$ en vez del único conjunto $\mathcal{A}$. La función de transición $\mathcal{T}$ es un operador que actúa sobre todas las combinaciones posibles entre el conjunto de estados y el espacio de acciones combinado de todos los agentes: $\mathcal{T} : \mathcal{S} \times \mathcal{A}_1 \times \mathcal{A}_2 \times ... \times \mathcal{A}_N \mapsto \mathcal{S}$. Similarmente, las funciones $\mathcal{R}_i$ definen las recompensas obtenidas por los agentes: $\mathcal{R}_i : \mathcal{S} \times \mathcal{A}_i \mapsto \mathbb{R}$. En el caso de un juego basado en un proceso parcialmente observado, la definición 2.12 incluye también las listas de conjuntos de observaciones $\mathcal{O}_1, ..., \mathcal{O}_N$ y de funciones de observación $\mathcal{Z}_1, ..., \mathcal{Z}_N$ [7].

$$\mathcal{M}_G = \langle \mathcal{S}, \mathcal{A}_1, ..., \mathcal{A}_N, \mathcal{T}, \mathcal{R}_1, ..., \mathcal{R}_N, \gamma, \rho_0 \rangle \tag{2.12}$$

2.2. Estado del arte

2.2.1. Mitigación automática de ciberataques

La abstracción, flexibilidad y programabilidad que presentan las infraestructuras basadas en el paradigma SDN han permitido el desarrollo de múltiples sistemas de detección de ataques y de despliegue automático de contramedidas.

En lo que a los sistemas de detección de intrusión y prevención respecta (IDPS, de sus siglas en inglés de *Intrusion Detection and Prevention Systems*), se han propuesto varios sistemas inteligentes capaces de decidir qué contramedidas tomar. Uno de los más completos es NICE (*Network Intrusion detection and Countermeasure sElection in virtual network systems*) [6], que integra la detección de máquinas virtuales infectadas en entornos de computación en la nube con el

despliegue automático de las contramedidas óptimas. Toda la infraestructura propuesta en NICE se basa en redes SDN que utilizan el protocolo OpenFlow. Cada máquina de la red presenta una vulnerabilidad registrada en la lista *Common Vulnerabilities and Exposures* (CVE) [27], y el ataque se representa mediante un grafo en el que cada nodo es el estado previo o la consecuencia del ataque de una de las máquinas. La selección de la respuesta ante un ataque se toma teniendo en cuenta la intrusividad y el coste de la misma, por lo que se busca que el impacto que tiene la propia contramedida sea el mínimo. La Tabla 2.3 recoge la intrusividad y el coste de las contramedidas propuestas por NICE.

#	Contramedida	Intrusividad	Coste
1	Reenviar el tráfico	3	3
2	Aislar el tráfico	4	2
3	Inspeccionar a fondo los paquetes[3]	3	3
4	Crear reglas de filtrado	1	2
5	Cambiar dirección MAC	2	1
6	Cambiar dirección IP	2	1
7	Bloquear puerto	4	1
8	Corregir el software	5	4
9	Poner en cuarentena	5	2
10	Reconfigurar la red	0	5
11	Cambiar la topología de la red	0	5

Tabla 2.3: Contramedidas propuestas por NICE [6].

El sistema SnortFlow [18] sigue las pautas marcadas por NICE, pero todas las contramedidas propuestas se basan únicamente en acciones realizadas sobre la propia red (como la redirección del tráfico o el bloqueo de un puerto) y se implementan mediante un controlador OpenFlow.

Por otro lado, el uso de técnicas de aprendizaje automático para la detección y mitigación automática de ataques DoS y DDoS en redes SDN ha sido un tema muy estudiado durante los últimos años [17]. Por ejemplo, el trabajo de C. Li et al. [29] utiliza técnicas de aprendizaje automático para detectar el tráfico proveniente de ataques DDoS para posteriomente bloquearlo utilizando un controlador

[3]La inspección a fondo de los paquetes (DPI, de sus siglas en inglés de *Deep Packet Inspection*) se basa en procesar el contenido de los datos transmitidos a través de una red antes de que llegue a su destino final, con la intención de bloquear, reenviar o guardar la información necesaria, entre otros [28].

OpenFlow. M. Campos y J. Martins [30] proponen desplegar contramedidas que se adecúan a las características del ataque y realizan experimentos en los que su implementación es capaz de reconocer el tipo de ataque, seleccionar la contramedida correspondiente y llevarla a cabo mediante un controlador OpenFlow.

Aparte de las propuestas centradas en un aspecto práctico de la implementación de las contramedidas, trabajos como el de K. Hammar y R. Stadler plantean utilizar juegos de Markov multiagente para buscar estrategias de defensa frente a ciberataques [7]. Concretamente, el juego se plantea como un juego de Markov parcialmente observado y los autores estudian diferentes escenarios en los que varían las habilidades del atacante y del defensor. En su juego, el atacante debe moverse a través de un grafo para llegar a una máquina objetivo, simulando un problema parecido a los planteados en los *Cyber Range* [31]. En dicho trabajo se utilizan los algoritmos model-free PPO [32] y REINFORCE [33] para entrenar al agente.

2.2.2. Algoritmos de aprendizaje por refuerzo y el algoritmo MuZero

Una manera clásica de evaluar los algoritmos de aprendizaje por refuerzo ha sido enfrentarlos a juegos como el ajedrez o el Go para intentar batir las marcas humanas y obtener resultados sobrehumanos.

Uno de los primeros hitos en este campo, aparte de los ordenadores especializados para ganar al ajedrez o al shogi [34], fue el algoritmo AlphaGo desarrollado por la empresa DeepMind, que consiguió ganar por primera vez a un jugador profesional de Go [35]. Una modificación de ese algoritmo llamada AlphaGo Zero [36] obtuvo resultados sobrehumanos jugando al Go y dio paso a su sucesor AlphaZero, que consiguió vencer a campeones mundiales de ajedrez, shogi y Go con solo 24 horas de entrenamiento en cada caso [37]. Para ello utilizaron 500 TPUs de primera generación para generar las partidas jugadas de forma autónoma y 64 TPUs de segunda generación para entrenar las redes neuronales.

Los algoritmos basados en modelos han superado en múltiples ocasiones a los humanos en juegos clásicos como las damas, el ajedrez, el Go o el póker, pero han

solido mostrar resultados pocos satisfactorios al enfrentarse a entornos complejos como los juegos de la consola Atari 2600. En ese tipo de entornos los algoritmos model-free obtienen resultados superiores [38, 39, 40], pero muestran rendimientos mucho peores de lo esperado en juegos de mesa como los anteriormente citados.

El algoritmo MuZero presentado por DeepMind en 2020 (un algoritmo model-based) cambió del todo el estado del arte. MuZero logra rendimientos comparables a los de otros modelos en juegos de Atari 2600, mientras mantiene resultados sobrehumanos en los juegos de mesa clásicos, tal y como hacen otros los algoritmos previos basados en modelos [8].

El funcionamiento del algoritmo es el siguiente: el modelo μ_θ de parámetros θ predice para cada $k = 1, ..., K$ pasos la política (Ecuación 2.13), la función evaluadora (Ecuación 2.14) y la recompensa inmediata (Ecuación 2.15) basándose en las observaciones pasadas $o_1, ..., o_t$ y en las acciones futuras $a_{t+1}, ..., a_{t+k}$.

$$\mathbf{p}_t^k \approx \pi(a_{t+k+1} \mid o_1, ..., o_t, a_{t+1}, ...a_{t+k}) \tag{2.13}$$

$$v_t^k \approx \mathbb{E}\left[u_{t+k+1} + \gamma u_{t+k+2} + ... \mid o_1, ..., o_t, a_{t+1}, ..., a_{t+k}\right] \tag{2.14}$$

$$r_t^k \approx u_{t+k} \tag{2.15}$$

donde u es la recompensa real obtenida, π es la política seguida a la hora de tomar las acciones y γ es el factor de descuento del entorno.

En cada paso t, el modelo se representa mediante una combinación de las funciones de *dinámica*, de *predicción* y de *representación*.

La función de dinámica (Ecuación 2.16) calcula, para cada paso hipotético $k = 1...K$ (dado en t), la recompensa inmediata r_t^k y el nuevo *estado interno* s_t^k. Ese estado interno no tiene que ser una copia del estado real del entorno, si no que debe ser capaz de predecir las políticas, los valores y las recompensas.

$$r_t^k, s_t^k = g_\theta(s_t^{k-1}, a_t^k) \tag{2.16}$$

La función de predicción es una función determinista que calcula la política y

el valor de un estado s_t^k:

$$\mathbf{p}_t^k, v_t^k = f_\theta(s_t^k) \tag{2.17}$$

donde el estado inicial s_t^0 se obtiene mediante la función de representación h_θ: $s_t^0 = h_\theta(o_1, ..., o_t)$.

Una vez obtenido el modelo, se utiliza un Árbol de búsqueda de Monte Carlo (MCTS, de sus siglas en inglés de *Monte Carlo tree search*) para obtener la política recomendada π_t y el valor estimado ν_t para así seleccionar la acción $a_{t+1} \sim \pi_t$.

El objetivo principal del algoritmo MuZero es minimizar la función de pérdida (*loss function*) definida como

$$l_t(\theta) = \sum_{k=0}^{K} l^r(u_{t+k}, r_t^k) + l^v(z_{t+k}, v_t^k) + l^p(\pi_{t+k}, \mathbf{p}_t^k) + c||\theta||^2 \tag{2.18}$$

donde l^r, l^v y l^p son las pérdidas de la recompensa, el valor y la política, respectivamente. El valor objetivo z_t se define como $z_t = u_{t+1} + \gamma u_{t+2} + ... + \gamma^{n-1} u_{t+n} + \gamma^n \nu_{t+n}$ y $c||\theta||^2$ es el término de regularización L2.

Capítulo 3

Propuesta de investigación

La propuesta principal de este trabajo es utilizar el algoritmo MuZero para entrenar un modelo que sea capaz de decidir qué contramedidas tomar ante la intrusión de un atacante en una red SDN, con la intención de mitigar el ataque y minimizar el número de máquinas comprometidas. La interacción entre el intruso y el defensor se ha representado mediante un juego de Markov parcialmente observado de suma zero, en el que ambos agentes realizan las acciones (algunas de ellas estocásticas) de forma secuencial. El modelo se ha integrado en un entorno realista de emulación de redes SDN, basado en Mininet [41]. Los switches (virtuales) desplegados con Mininet son compatibles con la especificación OpenFlow y aceptan su control desde un controlador externo. En nuestro entorno de trabajo hemos usado Ryu [42]. La Figura 3.1 resume los tres bloques desarrollados en este proyecto y la interfaces utilizadas para comunicarse entre ellos.

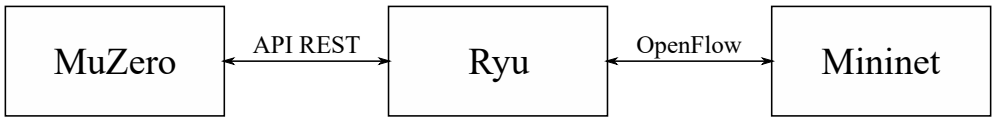


Figura 3.1: Diagrama de la propuesta realizada en este trabajo, en la que el modelo entrenado, el controlador de la red y el entorno de simulación son independientes.

A continuación se presentan los programas y librerias utilizados en la implementación, las decisiones referentes al diseño del juego y el entorno de red virtual implementado.

3.1. Herramientas utilizadas

- **Mininet [41]:** definición de redes virtuales, formadas por enlaces Ethernet, switches OpenFlow y hosts Linux conectados a los switches dentro de una sola máquina. La topología de la red, así como otros parámetros de switches, enlaces y hosts se puede especificar mediante línea de comandos o mediante scripts de Python.

- **Ryu [42]:** controlador OpenFlow. Ryu es un entorno para programar redes SDN basadas en OpenFlow. En particular, se puede integrar perfectamente con topologías definidas mediante Mininet, aunque su función principal es controlar switches físicos. El controlador se programa en Python (lenguaje en el que está escrito el propio Ryu), y cuenta con facilidades para ofrecer una interfaz norte basada en un API REST [16].

- **MuZero General [43]:** una implementación de MuZero basada en el pseudocódigo proporcionado en el artículo original [8] y escrita en Python. Dicha implementación ofrece un entorno adecuado para entrenar juegos personalizados escritos en Python y evaluar el rendimiento del modelo obtenido. En el apartado 3.1.1 se explica el funcionamiento básico de la implementación.

En consecuencia, este proyecto se ha desarrollado íntegramente en Python [44]. La Tabla 3.1 muestra las versiones de las librerías o paquetes utilizados.

Paquete	Versión
eventlet	0.30.2
gym [45]	0.17.3
imageio [46]	2.9.0
mininet [41]	2.3.0
networkx [47]	2.5.1
nevergrad [48]	0.4.2post2
numpy [49]	1.16.4
nvidia-smi	450.119.03
paramiko	2.7.2
ray [50]	1.2.0
ryu [42]	4.34
seaborn [51]	0.11.0
tensorboard [52]	2.4.0
torch [53]	1.8.1

Tabla 3.1: Versiones de las librerias más relevantes.

3.1.1. Uso básico de MuZero General

El código de la implementación de MuZero utilizada en este proyecto puede encontrarse en GitHub[1]. Una vez descargado, basta con instalar los paquetes enumerados en el fichero `requirements.txt` para tener listo el entorno en el que se ejecutará MuZero. Adicionalmente, si se quiere hacer uso de GPUs para entrenar el modelo, será necesario instalar los drivers adecuados.

MuZero se inicia al ejecutar el archivo `muzero.py`—que se encuentra en la carpeta principal del repositorio—con alguna versión igual o superior a Python 3.6. Una vez ejecutado, en la consola de comandos se nos pedirá que seleccionemos el juego con el que queremos trabajar y aparecerá un menú parecido a este:

```
0. Train
1. Load pretrained model
2. Diagnose model
3. Render some self play games
4. Play against MuZero
5. Test the game manually
6. Hyperparameter search
7. Exit
```

Las opciones principales son las que permiten entrenar un modelo (`Train`), cargar un modelo previamente entrenado (`Load pretrained model`) y probar el juego manualmente (`Test the game manually`). Al cargar un juego y entrenar el modelo por un número determinado de pasos (los pasos a realizar se especifican en la configuración del propio juego) se obtienen unos ficheros que pueden ser cargados posteriormente. Una ver cargado el modelo entrenado, se pueden visualizar partidas en las que MuZero controla a todos los agentes (opción 3) o enfrentarse manualmente a él (mediante la opción 4, solo disponible en juegos de dos jugadores). Cabe mencionar que la opción de probar el juego de forma manual es especialmente útil durante el desarrollo del juego.

[1] https://github.com/werner-duvaud/muzero-general

3.2. Diseño del juego

En este apartado se explican los diferentes elementos diseñados para obtener un juego de Markov parcialmente observado que represente la interacción entre el atacante y el defensor.

3.2.1. Los agentes y el entorno

El juego está planteado como una partida entre dos jugadores, el atacante y el defensor, en la que el atacante trata de comprometer objetivos clave de una red y el defensor intenta mitigar el ataque sufrido.

Aunque el algoritmo MuZero se diseñó para enfrentarse a juegos de mesa o videojuegos, como en este caso el entorno que se quiere simular está basado en una red de ordenadores, se ha optado por utilizar un grafo para representarla. A efectos prácticos, MuZero puede utilizarse para entrenar agentes en cualquier entorno que pueda representarse mediante un vector o una matriz. La relación entre el grafo y la matriz con la que juega MuZero se explica en el apartado 3.2.2.

En este juego, la red atacada está compuesta por N nodos vulnerables, de los cuales m forman parte de una *honeynet*. Una honeynet es una parte aislada de la red en la que las máquinas (llamadas *honeypots*) se utilizan como trampas para los atacantes [54]. Esas máquinas no proveen ningún servicio real, pero son vulnerables a los ataques, haciendo que los atacantes pierdan tiempo y recursos explorándolas y permitiendo a los defensores obtener información de los intrusos [55]. Esta estructura es similar a la propuesta por el proyecto Science DMZ [56].

Similarmente a lo propuesto por el proyecto NICE, a cada nodo de la red (también llamados *hosts*) se le asigna una vulnerabilidad. Tal y como hace la *National Vulnerability Database* (NVD) [57], una vulnerabilidad se evalúa cuantitativamente mediante su Puntuación Base (BS, de sus siglas en inglés de *Base Score*). El valor de la Puntuación Base depende de la Sub-Puntuación de Impacto (ISS, de sus siglas en inglés de *Impact Sub-Score*), de la Explotabilidad (*Exploitability*) y del Impacto (*Impact*). De acuerdo con la versión 3.1 del estándar de evaluación de vulnerabilidades *Common Vulnerability Scoring System* (CVSS)

[58], la Sub-Puntuación de Impacto se define como

$$ISS = 1 - [(1 - C) \times (1 - I) \times (1 - A)] \tag{3.1}$$

donde C es el *Confidentiality Impact*, I es el *Integrity Impact* y A es el *Availability Impact*.

La Explotabilidad de un host depende de los factores *Attack Vector* (AV), *Attack Complexity* (AC), *Priviledges Required* (PR) y *User Interaction* (UI), tal y como muestra la Ecuación 3.2.

$$\text{Explotabilidad} = 8{,}22 \times AV \times AC \times PR \times UI \tag{3.2}$$

El Impacto de la vulnerabilidad depende del *Scope* (S) de la misma. Dicho factor determina si el host afectado es el mismo que presenta la vulnerabilidad (*Unchanged*) u otro diferente (*Changed*). Cuando el Scope es Changed, el Impacto se calcula según

$$\text{Impacto} = 6{,}42 \times ISS \tag{3.3}$$

mientras que si es Unchanged,

$$\text{Impacto} = 7{,}52 \times (ISS - 0{,}029) - 3{,}25 \times (ISS - 0{,}02)^{15}. \tag{3.4}$$

Finalmente, la puntuación se calcula de formas diferentes dependiendo de los valores del Impacto y del Scope. Si el Impacto $= 0$, $BS = 0$. Si Impacto > 0, BS se calcula como

$$BS = \text{roundup}\left(\min\left[1{,}08 \times (\text{Impacto} + \text{Explotabilidad}), 10\right]\right) \tag{3.5}$$

si el Scope es Changed, y como

$$BS = \text{roundup}\left(\min\left[(\text{Impacto} + \text{Explotabilidad}), 10\right]\right) \tag{3.6}$$

si el Scope es Unchanged. La función roundup devuelve el número más pequeño, teniendo en cuenta un decimal, que es mayor o igual al valor de entrada [58].

En lo que al juego respecta, el Scope de una vulnerabilidad determina qué acciones puede llevar a cabo el atacante si consigue explotarla: si el Scope es Unchanged, el atacante sólo podría leer archivos de la máquina objetivo, mientras que si el Scope es Changed el atacante obtendría permisos de ejecución y podría explorar y explotar las máquinas a las que está conectado el host que presenta la vulnerabilidad.

La red inicial consta, por lo tanto, de $N - m$ hosts conectados entre ellos formando una red lógica determinada (ver Sección 3.4) y m máquinas aisladas de la red principal pero conectadas entre ellas. A cada nodo se le asigna una vulnerabilidad y puede estar en tres estados diferentes—normal (estado 0), explorado (estado 1) y atacado (estado 2)—, permitiendo únicamente las transiciones $0 \rightarrow 1 \rightarrow 2$. Para mantener un número reducido de acciones posibles se ha decidido que las máquinas no pueden volver a un estado anterior. Al inicio del juego una de las máquinas de la red principal se establece como *bandera* (estado -1) y el objetivo principal del atacante será atacar esa máquina. Otro de los hosts con Scope Changed se sitúa en el estado 2, permitiendo empezar el ataque desde ahí.

Con el fin de ilustrar las explicaciones que vienen a continuación, en el resto del informe se representará el estado de la red mediante grafos, ya que también se utilizan para mostrar el estado del juego. Formalmente, un grafo $G = (V, E)$ es una pareja ordenada donde V es un conjunto de nodos o vértices y E un conjunto de arcos o aristas tal que $E \subseteq \{\{x, y\} \mid x, y \in V \wedge x \neq y\}$. A la hora de representar el sistema, cada nodo del grafo representa un host y los arcos muestran que el tráfico entre dos máquinas no está bloqueado. Como cada host puede estar en diferentes estados, se ha optado por una codificación de colores para representarlos en las figuras (ver Figura 3.2). En dicha codificación, la situación inicial de las máquinas se representa mediante nodos de fondo blanco, las máquinas exploradas con nodos amarillos y las atacadas mediante el color rojo. El nodo con el borde verde señala que esa máquina es la bandera. La etiqueta puesta en cada nodo solo es un identificador y no tiene relevancia desde el punto de vista del juego.

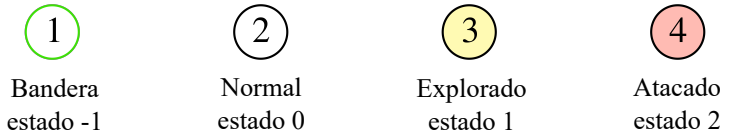


Figura 3.2: Codificación utilizada para representar los diferentes estados en los que pueden estar los nodos.

A modo de ejemplo, la Tabla 3.2 recoge un posible entorno formado por $N = 12$ nodos (con $m = 4$ nodos formando una honeynet) y la Figura 3.3 muestra dicha red mediante un grafo. Cabe subrayar que dicho grafo no representa la topología real (compuesta por switches, hosts y conexiones Ethernet) de la red, puesto que un arco entre dos nodos no representa una conexión física, si no que esos nodos pueden intercambiarse paquetes.

Nodo	*BS*	Expl.	Imp.	Scope	Vecinos
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

Tabla 3.2: Ejemplo de entorno en el que se desarrolla el juego.

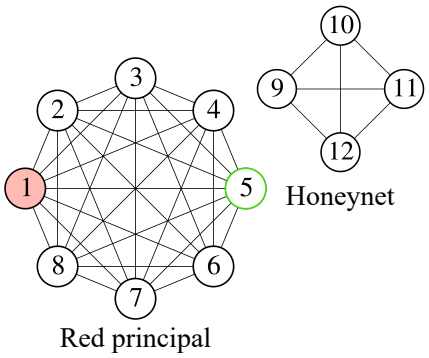


Figura 3.3: Representación gráfica del entorno descrito por la Tabla 3.2.

3.2.2. Las observaciones

El desarrollo de la intrusión y su mitigación se plantean como procesos de decisión parcialmente observados, ya que ni el atacante ni el defensor tienen la información total del estado del juego. Para representar esta circunstancia, para cada estado del juego se obtienen tres grafos diferentes: el grafo general $G_G = (V_G, E_G)$ que representa el estado completo de la red, el del atacante $G_A = (V_A, E_A)$ que es la vista parcial que tiene del estado y el grafo del defensor $G_D = (V_D, E_D)$ que también ofrece solo parte de la información.

Teniendo en cuenta que MuZero necesita una representación matricial del entorno, es necesario obtener una matriz equivalente del estado del juego. Aunque la matriz de adyacencia permite representar las conexiones existentes en un grafo, no es suficiente para describirlo en su totalidad, puesto que cada nodo puede estar en diferentes estados. Aprovechando que la diagonal principal de dicha matriz es nula, se genera una matriz diagonal que representa el estado en el que se encuentra cada nodo según la codificación descrita en el apartado anterior. Finalmente, el tablero observado por cada jugador es una matriz que se obtiene sumando las dos anteriores. Por lo tanto, los grafos G_G, G_A y G_D se representan mediante unas matrices M_G, M_A y M_D respectivamente, siguiendo el proceso descrito. La Figura 3.4 resume la obtención de la matriz para un grafo sencillo con tres nodos.

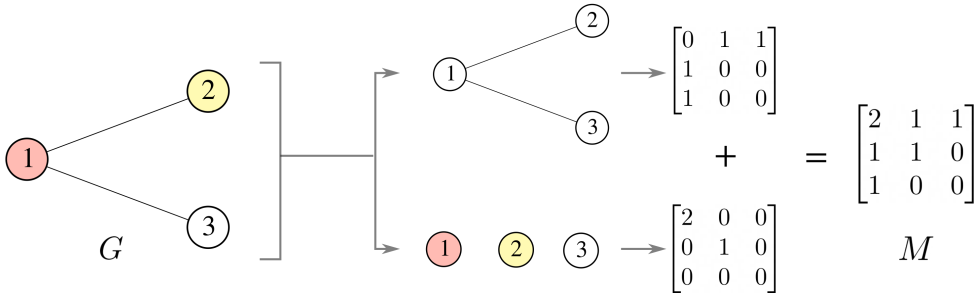
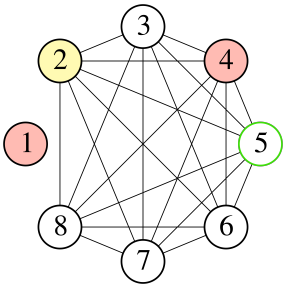


Figura 3.4: Obtención de la matriz M que representa el estado observado mediante el grafo G.

En general, todos los grafos son diferentes entre sí, pues las acciones llevadas a cabo por los jugadores (explicadas en el apartado 3.2.3) solo tienen efecto en el grafo general y en el del jugador que la ha realizado. En consecuencia, cada jugador solo tiene una observación parcial del juego. La Figura 3.5 muestra los tres grafos calculados de un mismo estado y la matriz recibida por cada jugador. El estado real es el siguiente: partiendo de la situación inicial determinada por la Tabla 3.2, el atacante ha atacado la máquina 4 y ha explorado la máquina 2. En realidad, la máquina 1 está aislada del resto (tal y como se ve en los grafos G_G y G_D) pero el atacante aún no se ha percatado del cambio que ha habido en la conectividad entre los nodos de la red. Similarmente, el defensor ha detectado que la máquina 1 ha sido atacada (y podemos suponer que ha tomado alguna de las medidas explicadas en el apartado 3.2.3) pero no percibe que el nodo 2 está en el estado 1 y el nodo 4 en el estado 2.

Estado real

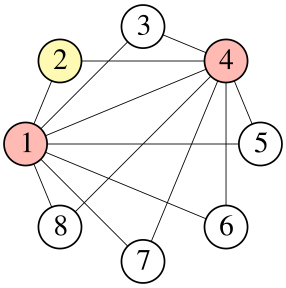


G_G

$$\begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 2 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & -1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

M_G

Observación del atacante

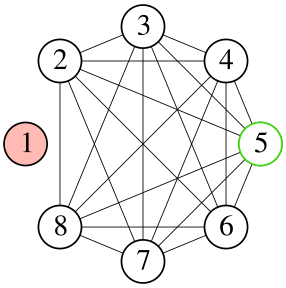


G_A

$$\begin{bmatrix} 2 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 2 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

M_A

Observación del defensor



G_D

$$\begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & -1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

M_D

Figura 3.5: Ejemplo de las observaciones obtenidas por el defensor y el atacante de un mismo estado.

3.2.3. Las acciones

A la hora de diseñar las acciones disponibles para cada jugador, se han asumido las siguientes condiciones:

- Solo se consideran las acciones defensivas que pueden implementarse utilizando un controlador SDN, quedando fuera del estudio opciones como actualizaciones de software o cambios en la topología de la infraestructura como añadir o quitar un switch o mover físicamente un host.
- Solo se consideran las acciones ofensivas que se realizan contra las máquinas conectadas a los elementos del plano de datos, siendo esos últimos completamente invulnerables. El controlador SDN tampoco puede ser el objetivo de un ataque.
- El algoritmo MuZero necesita que el tamaño del tablero se mantenga constante a lo largo del juego, por lo que el número de nodos no puede variar: no se pueden desplegar máquinas nuevas ni eliminarlas completamente.

Merece la pena insistir en que la toma de decisiones se realiza teniendo en cuenta la observación recibida por el agente y no basándose en el estado real del juego. Por lo tanto, las acciones *legales* que en teoría puede realizar un agente pueden no ser factibles en la práctica o puede que no tengan el efecto esperado debido a las diferencias entre la observación del agente y el estado real. Además, como ya se ha mencionado, se trata de un juego estocástico, por lo que hay acciones que tienen un factor probabilístico. Por ejemplo, que el atacante consiga o no explotar una vulnerabilidad de una máquina, depende directamente de las características de la vulnerabilidad, por lo que no siempre obtendrá el resultado esperado.

El abanico de acciones disponibles para el defensor es el siguiente:

- **Verificar estado**

  El defensor verifica secuencialmente si el estado de todos los nodos se corresponde con el de su observación. Inspirado en NICE, la probabilidad de detectar un nodo comprometido es proporcional a la Puntuación Base de la vulnerabilidad de la máquina atacada.

- **Aislar nodo**

  Acción determinista que aísla completamente a una máquina atacada.

- **Mandar nodo a honeynet**

  Acción determinista que bloquea todo el tráfico entre la máquina afectada y la red principal y permite la comunicación del nodo con el resto de nodos en la honeynet (ver Figura 3.6). En nuestro juego, el objetivo de esta acción es hacer que el defensor gane tiempo para verificar el estado del resto de la red mientras el atacante explora y ataca su entorno. Los ataques realizados a los miembros de la honeynet no afectan al funcionamiento de la red real.

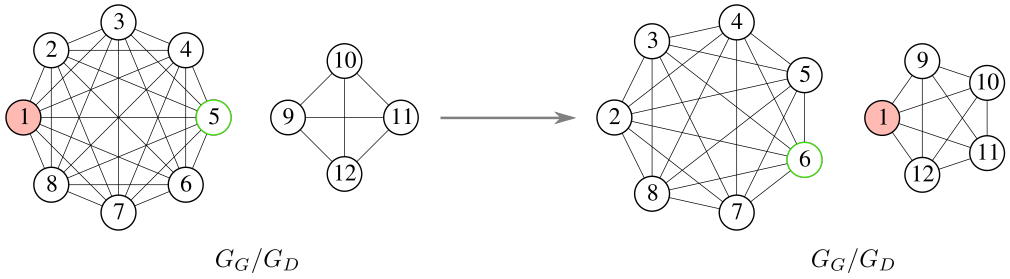


Figura 3.6: Estado de la red principal y de la honeynet antes de redirigir todo el tráfico del nodo 1 a los nodos de la honeynet (izquierda) y después (derecha).

- **Mover la bandera**

  Esta acción simula la migración de los servicios críticos de una máquina a otra. El defensor puede intentar llevar a cabo esta acción si la máquina a la que quiere mover la bandera no está comprometida en su observación. Aunque es una acción determinista, si la máquina ha sido atacada (el estado real de la máquina es 2 en vez del estado observado 0), no se puede realizar la acción pero el defensor puede detectar que la máquina ha sido atacada con un probabilidad igual a la utilizada en la acción *Verificar estado*[2]. No se considera la migración de servicios a múltiples máquinas (división de la bandera).

A cada contramedida se le ha establecido un valor que representa el coste total de llevar a cabo esa acción. Aunque las contramedidas propuestas en NICE [6] y

[2] Esta decisión puede parecer poco realista, pero si al defensor se le permite detectar directamente que una máquina está infectada cuando trata de mover ahí la bandera, el defensor utilizará esta acción para detectar a los intrusos, ya que es más efectiva que verificar el estado de la red.

las planteadas en este trabajo no son idénticas, sí que son equivalentes, por lo que a cada acción se le ha asignado el coste y la intrusividad de su medida equivalente mostrada en la Tabla 2.3. La acción *Verificar estado* no tiene equivalente en NICE, pues no se trata de una contramedida si no de una tarea de monitorización, por lo que se le ha asignado un coste bajo y una intrusividad nula. El coste total de cada medida, que tiene en cuenta el efecto que tiene en la red y los recursos necesarios para implementarla, se obtiene sumando el coste y la intrusividad de la misma. La Tabla 3.3 recoge los costes de las acciones y sus medidas equivalentes.

Acción del defensor	Contramedida en NICE	Intrusividad	Coste	Coste total
Verificar estado	—	0	1	1
Aislar nodo	Aislar el tráfico	4	2	6
Mandar nodo a honeynet	Poner en cuarentena	5	2	7
Mover la bandera	Reconfigurar la red	0	5	5

Tabla 3.3: Costes de las acciones del defensor y su relación con las contramedidas propuestas en NICE [6].

El objetivo de definir estos costes es hacer que el defensor mitigue el ataque siguiendo un estrategia óptima, minimizando así los recursos utilizados.

En cuanto al atacante, las acciones disponibles son las siguientes:

- **Explorar la topología**

  El atacante obtiene la lista de máquinas alcanzables desde máquinas ya atacadas y con Scope Changed.

- **Explorar vulnerabilidades**

  Esta acción explora la vulnerabilidad de una máquina desde otra con Scope Changed. Si la acción es viable, la probabilidad de detectar una vulnerabilidad es Explotabilidad/10, que toma valores entre 0 y 1. Explorar la vulnerabilidad hace que el estado del nodo cambie de 0 a 1.

- **Atacar vulnerabilidad**

  El atacante intenta explotar una de las máquinas vulnerables desde otra con Scope Changed. De nuevo, la probabilidad de comprometer una máquina es una décima parte de la Explotabilidad de su vulnerabilidad. Al comprometer un nodo, su estado pasa de 1 a 2.

Una vez presentadas todas las acciones, es pertinente retomar la siguiente idea: que el atacante decida llevar a cabo una acción no significa que tenga éxito al realizarla. El pseudocódigo presentado en el Algoritmo 1 muestra cómo se ejecuta una acción para el caso del atacante. A la hora de llevar a cabo una exploración o un ataque, el intruso toma el conjunto de nodos vecinos de la máquina objetivo $n_O \in V_A$ del grafo G_A. De ese conjunto toma aleatoriamente uno de los nodos n_A en el que tiene permisos de ejecución y trata de realizar la acción. Si el arco entre n_O y n_A no existe en G_G, el atacante no puede utilizar dicha conexión para atacar, por lo que elimina el arco de su grafo y selecciona otro nodo. Si el arco existe, se genera un número aleatorio entre 0 y 1 y la Explotabilidad de la máquina objetivo se divide entre 10, escalándola así a ese rango. De esta manera, la acción puede llevarse a cabo si el número aleatorio es menor que la Explotabilidad escalada, por lo que la probabilidad de éxito es de Explotabilidad/10.

Algoritmo 1: Desarrollo de un ataque

```
action, objective_node ← select_action(observation);
neighbours ← get_neighbours(objective_node, G_A);
action_possible ← false;
for neighbour ∈ neighbours do
    if edge(objective_node, neighbour) ∈ E_G then
        action_possible ← true;
        break;
    else
        remove_edge(objective_node, neighbour, G_A );
    end
end
if action_possible == true then
    exploitability ← get_exploitability(objective_node);
    random_number ← get_random_number(0,1);
    if exploitability/10 ≥ random_number then
        apply_action(action, objective_node);
    end
end
```

Supongamos el caso trivial en el que el grafo general y el del atacante son los mostrados en la Figura 3.7. En ese estado, (1) el atacante decide intentar explotar la vulnerabilidad que ha detectado en la máquina 2 puesto que, según su observación, el host 1 en el que tiene permisos de ejecución está conectado a ella. En realidad, (2) el tráfico entre ambas máquinas está bloqueado, por lo que cuando el atacante intenta realizar la acción esta no tiene el efecto esperado, pero (3) actualiza la observación que tiene del entorno, aunque esta aún no se corresponde al estado real (4).

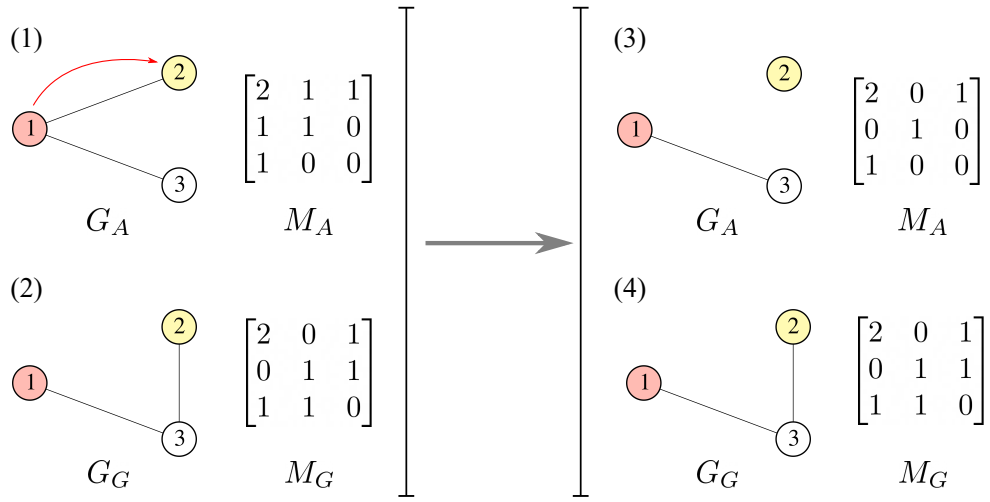


Figura 3.7: Ejemplo de las situaciones derivadas de las diferencias entre el estado real representado por G_G y el observado por el atacante.

Aunque las acciones del atacante no tienen un coste establecido, el hecho de que no sean deterministas y que, además, la probabilidad de explotar las vulnerabilidades sea proporcional a la Explotabilidad hace que dichas acciones tengan un coste implícito.

3.2.4. Los objetivos

El objetivo principal del atacante es atacar la máquina bandera y el del defensor es evitar que eso ocurra. Para ello, ambos jugadores van realizando acciones por turnos hasta que el atacante llega a la bandera (gana el atacante) o no puede hacer ninguna acción más (gana el defensor).

3.2.5. Las recompensas

La implementación de MuZero utilizada plantea los juegos de dos jugadores como juegos de suma cero, cambiando automáticamente el signo de la recompensa del adversario. Por lo tanto, cada jugador, además de recibir la recompensa que le corresponde, también es penalizado con la recompensa obtenida por el adversario. La finalidad de cada agente es maximizar su recompensa mientras minimiza la de su adversario, por lo que los objetivos secundarios más allá de ganar la partida se especifican mediante el diseño de las recompensas.

Aunque en este trabajo se han probado diferentes tipos de recompensas, en todas ellas se recompensa únicamente la acción que consigue que el agente que la realiza gane la partida[3], dando una recompensa nula al resto de acciones. En una primera implementación se ha optado por un sistema de recompensas simple que recompensa al atacante con un valor proporcional a las máquinas afectadas. Para ello se calcula la traza de la observación general y, mientras que al atacante se le premia directamente con ese valor, al defensor se le recompensa con la puntuación máxima que puede obtener el atacante (2× tamaño del tablero) menos la recompensa del atacante, tal y como se muestra en la Ecuación 3.7.

$$r_t = \begin{cases} \mathrm{Tr}\left(M_G\right), & \text{vencedor = atacante} \\ \max\left[0,\ 2N - \mathrm{Tr}\left(M_G\right)\right], & \text{vencedor = defensor} \\ 0, & \text{en el resto de los casos} \end{cases} \tag{3.7}$$

siendo N el número de nodos en la red (y el tamaño de la matriz M_G) y $\mathrm{Tr}(M_G)$ la traza de la matriz M_G.

Hay que tener en cuenta que el juego también termina tras un número determinado de jugadas, evitando así que haya partidas infinitas. Las partidas que finalizan por ese criterio se consideran empate y ninguno de los agentes recibe una recompensa. Este sistema de recompensas no considera el coste de las contramedidas ni el impacto de los ataques, por lo que la estrategia desarrollada por el defensor no tiene por qué ser la óptima en lo que a recursos utilizados respecta.

[3] Al igual que en otros juegos como el ajedrez, en este caso los agentes solo pueden ganar la partida tras una acción propia, no tras la acción del adversario.

Seguidamente, con el objetivo de entrenar a un defensor capaz de minimizar el esfuerzo requerido para mitigar la intrusión, se han implementado las recompensas mostradas en la Ecuación 3.8,

$$r_t = \begin{cases} IT, & \text{vencedor} = \text{atacante} \\ \max(0, JM - IT - CT), & \text{vencedor} = \text{defensor} \\ 0, & \text{en el resto de los casos} \end{cases} \tag{3.8}$$

donde IT es el impacto total causado por el atacante, JM es el máximo de jugadas permitidas antes de terminar la partida y CT es el coste total de las contramedidas implementadas.

El impacto total es la suma del Impacto de las vulnerabilidades explotadas multiplicado por 10, haciendo así que sea del mismo orden de magnitud que el coste total de las contramedidas y, en consecuencia, de la recompensa obtenida por el defensor. El coste total se calcula sumando los costes de todas las contramedidas implementadas por el defensor, utilizando los valores recogidos en la Tabla 3.3. Este sistema de recompensas premia que el agente trate de finalizar la partida y también evalúa de manera positiva que el atacante comprometa el máximo número de nodos posible, incentivando así que explore el máximo posible de la red.

Merece la pena hacer un par de comentarios sobre el diseño de las recompensas y el efecto que tiene en el entrenamiento del modelo y en los resultados. Estas consideraciones solo deberían aplicarse a juegos similares al planteado en este trabajo, siendo muy difícil establecer unas pautas aplicables a todos los problemas.

Por un lado, podría pensarse que la mejor estrategia es la de recompensar a los agentes tras cada acción en vez de esperar a finalizar la partida. Si siguiendo esa estrategia se recompensan las máquinas afectadas o las contramedidas realizadas, los agentes tienden a "colaborar" entre ellos y a realizar partidas en las que ambos obtienen una cierta cantidad de puntos antes de tratar de ganar. Por lo tanto, es más adecuado dar una única recompensa al vencedor de la partida, tal y como se hace cuando se trabaja con juegos como el ajedrez [8].

Por el otro, también parece lógico que cuando el defensor gana la partida, se le recompense con el daño causado por el atacante con el signo cambiado. Así, a mayor impacto, menor sería la recompensa obtenida por el defensor. Este sistema de recompensas haría que la recompensa máxima obtenida por el defensor fuese 0. Como la recompensa solo se obtiene al final y las partidas tienen un número finito de pasos, cuando hay un empate ninguno de los agentes consigue ningún punto. Un entrenamiento con ese diseño podría dar lugar a unos agentes "apáticos", ya que el defensor obtendría su recompensa máxima sin realizar ninguna acción que trate de mitigar el ataque (revisando todo el tiempo el estado de las máquinas, por ejemplo).

Además, recompensar negativamente a uno de los agentes puede tener un efecto indeseado. Supongamos, por ejemplo, que utilizamos el sistema de recompensas de la Ecuación 3.8 pero sin evitar que el defensor obtenga recompensas negativas. En determinados casos, por ejemplo si el atacante se limita a observar la topología de la red, el defensor podría recibir una recompensa negativa fruto únicamente del coste de las contramedidas implementadas. Como MuZero cambia automáticamente el signo de las recompensas para evaluar al agente que ha perdido la partida, eso supondría que el atacante sería recompensado positivamente aunque no hubiese realizado ningún ataque activo, por lo que podría derivar en un atacante inactivo.

Cabe mencionar que, probablemente, con una sintonización adecuada de MuZero y con una capacidad de cómputo mayor (el hardware utilizado se explica en el Capítulo 4) los modelos entrenados con recompensas diferentes llegarían a desarrollar estrategias similares.

3.3. Implementación del juego

Una vez explicadas las características del juego y su funcionamiento, es interesante dar una idea general de cómo se implementa utilizando Python. Los juegos se definen como objetos `Env` de OpenAI Gym [45], un *toolkit* que permite desarrollar y comparar algoritmos de aprendizaje por refuerzo. Gym permite

desacoplar la especificación del juego del algoritmo de entrenamiento, por lo que el juego desarrollado en este proyecto podría entrenarse con otros algoritmos.

Los métodos básicos para definir un juego mediante un objeto `Env` son los siguientes:

- `reset(self)`: reinicia el entorno a su estado inicial y devuelve una observación.
- `step(self, action)`: lleva a cabo la acción seleccionada (por el usuario o por el algoritmo) y devuelve una observación, la recompensa y si la partida ha finalizado o no.
- `render(self)`: muestra el estado del juego.

Adicionalmente, se han definido estos métodos que simplifican el desarrollo de los ya mencionados:

- `to_play(self)`: devuelve el identificador del jugador al que le toca jugar.
- `get_observation(self)`: devuelve las matrices de las observaciones de los agentes y la del estado real.
- `legal_actions(self)`: devuelve la lista de acciones que puede realizar el agente basándose en la observación que tiene del entorno.
- `is_finished(self)`: devuelve si el juego ha terminado o no y el identificador del ganador.
- `get_reward(self, done, winner)`: devuelve la recompensa correspondiente al sistema de recompensas escogido. La variable `done` determina si la partida ha finalizado y `winner` es el identificador del ganador.
- `expert_action(self)`: permite definir la estrategia llevada a cabo por un agente experto. Devuelve la acción a realizar por el agente.

3.4. Infraestructura

A diferencia del trabajo realizado por K. Hammar Y R. Stadler [7], el segundo objetivo de este proyecto es implementar un sistema que realmente pueda llevar a cabo las contramedidas en una red SDN. Como ya se ha mencionado antes, la red SDN se ha emulado en Mininet y como controlador se ha utilizado Ryu.

Esta infraestructura tiene dos objetivos principales:

- Servir de puente entre el juego púramente teórico y su aplicación en redes SDN reales.
- Ser la base de implementaciones más complejas obtenidas como resultado de reemplazar o ampliar las funcionalidades de cualquiera de los bloques—el juego, el controlador o el entorno de simulación—de manera independiente.

La configuración de la topología de la red y de las conexiones entre los hosts se realiza mediante dos ficheros JSON: `topology.json` y `graph.json`. El primero determina los switches OpenFlow utilizados, el número de hosts desplegados y a qué switch está conectado cada uno de ellos. El segundo fichero determina las características de cada host: las direcciones IP y MAC, los hosts con los que puede comunicarse y la vulnerabilidad que presenta cada nodo. Aparte de las cuestiones relacionadas con la red, en dicho fichero se especifica también qué nodos están inicialmente atacados y cual de ellos es la bandera.

En este trabajo, la instalación de Mininet se ha realizado en una máquina virtual, manteniéndola separada del controlador y del modelo en sí. En consecuencia, para desplegar la red con la topología especificada en los archivos de configuración, se ha creado un script de Python que toma como variables de entrada las cadenas de caracteres correspondientes al contenido de los archivos JSON y utiliza la API de Python que ofrece Mininet para iniciar los switches y los hosts virtuales. De esta manera, aparte de poder iniciar el entorno de forma manual, también puede desplegarse desde el juego utilizando una conexión SSH.

El funcionamiento básico de los switches se basa en este ejemplo [59] proporcionado por los desarrolladores de Ryu: cuando a un switch llega un paquete

que no se empareja con ningún flujo, el switch envía una copia de dicho paquete al controlador. El controlador entonces ordena (1) realizar una "inundación" o envío de una copia del paquete por todos los puertos, excepto aquel por el que fue recibido y (2) instalar en los switches los flujos nuevos que sean necesarios para que en el futuro el reenvío sea directo. El resultado de esta forma de trabajar es que en la red se permite la comunicación de todos los nodos con todos. Aunque inicialmente esa comunicación es poco eficiente porque tiene que intervenir el controlador y realizar la inundación, paulatinamente se van configurando los switches, de forma que los siguientes reenvíos se realizan directamente. Una vez inicializada la red, se instalan flujos para que la conectividad de la red sea la determinada en los archivos de configuración y, paralelamente, se crean los grafos correspondientes al juego. Cabe mencionar que diferentes topologías de red pueden dar lugar a un mismo grafo de juego, ya que estos grafos no dependen de la topología de la red, si no de la conectividad.

Mientras que las contramedidas han sido implementadas mediante Ryu, los ataques planteados en el juego deben entenderse como abstracciones de las acciones que llevaría a cabo un atacante experto y, por lo tanto, no han sido implementadas en el entorno de simulación. A continuación se muestra cómo ejecuta el controlador las medidas seleccionadas por MuZero:

- **Aislar nodo**

  El controlador instala en los switches flujos que bloquean todos los paquetes con origen o destino a la IP de la máquina aislada.

- **Mandar nodo a honeynet**

  Se selecciona un switch de una honeynet aleatoriamente y el controlador elimina todos los flujos que bloqueaban el tráfico entre los nodos de esa honeynet y la máquina objetivo. Además, se instalan flujos en los switches de la red principal que bloquean el tráfico entrante y saliente de esa máquina.

- **Verificar estado**

  Esta acción no se ha implementado en el controlador, pero en el futuro podría añadirse un agente que estudiase el estado de la red y detectase intrusos, utilizando herramientas como un sistema de gestión de eventos e

información de seguridad (SIEM, de sus siglas en inglés de *Security Information and Event Management*) [60].

- **Mover la bandera**

  No se implementa mediante el controlador puesto que no es capaz de cambiar las direcciones IP y MAC de un host—esto podría utilizarse para mover la bandera de una máquina a otra sin realmente migrar los servicios. En futuras implementaciones el controlador redirigirá el tráfico a la nueva dirección una vez se haya cambiado.

3.5. Desarrollo del juego

La Figura 3.8 muestra el despliegue y el desarrollo del juego. Este flujo se repite cada vez que MuZero comienza una partida. En dicha figura puede verse cómo la red implementada en Mininet se utiliza para proyectar en ella las contramedidas aplicadas por el defensor. A la hora de entrenar el modelo, el juego se desarrolla íntegramente en los tableros, pero opcionalmente puede hacerse que MuZero interactúe directamente con el controlador SDN.

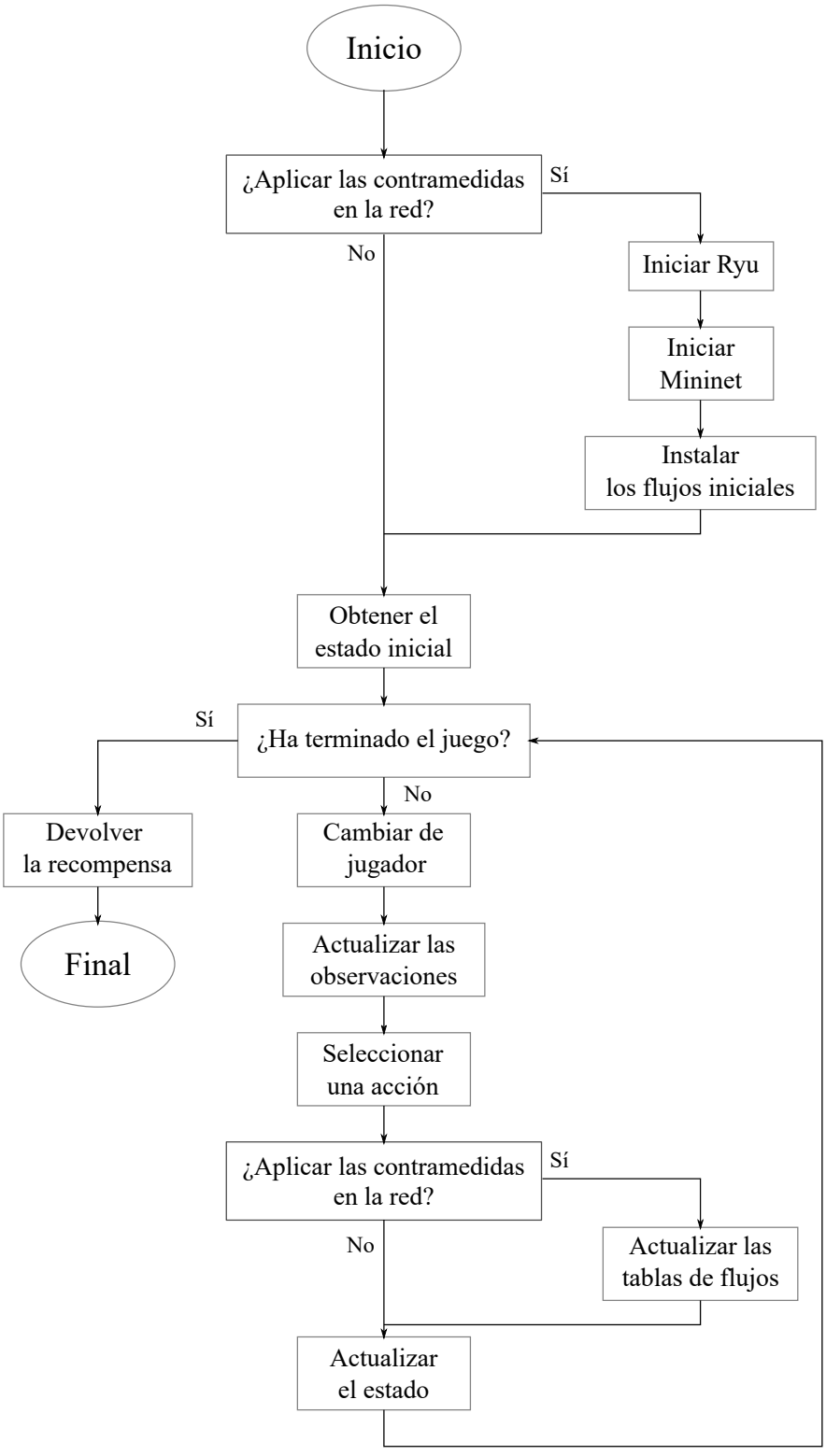


Figura 3.8: Diagrama del despliegue y desarrollo del juego.

Capítulo 4

Diseño experimental y resultados

En este capítulo se presentan los experimentos realizados y los resultados obtenidos en cada uno de ellos. Los objetivos de los experimentos son los siguientes:

1. Demostrar que las estrategias del defensor y del atacante mejoran a medida que se va entrenando el modelo.
2. Demostrar que el juego presentado en el apartado anterior puede adaptarse a diferentes redes.

En líneas generales, los experimentos se basan en establecer unas condiciones del juego, entrenar el modelo utilizando MuZero y realizar diferentes pruebas para medir el desarrollo de las estrategias de los agentes. Como métricas del desempeño de los jugadores se han considerado el número de jugadas que necesitan los jugadores para ganar la partida—la longitud de los episodios—, las recompensas obtenidas por los jugadores y el porcentaje de partidas ganadas por cada uno de ellos. Durante los experimentos se han probado diferentes configuraciones del juego permitiendo refinar el diseño del propio juego.

Aunque las peculiaridades de cada experimento se explican en los siguientes apartados, es interesante dar una visión global del desarrollo de las diferentes pruebas. Se han estudiado dos casos de uso basados en redes lógicas e infraestructuras diferentes. El primero de ellos es una red "simple" compuesta por $N = 12$ nodos. Ocho de ellos forman la red principal y los otros cuatro componen una honeynet que, inicialmente, está aislada del resto de hosts, tal y como se muestra en la parte inferior de la Figura 4.1.

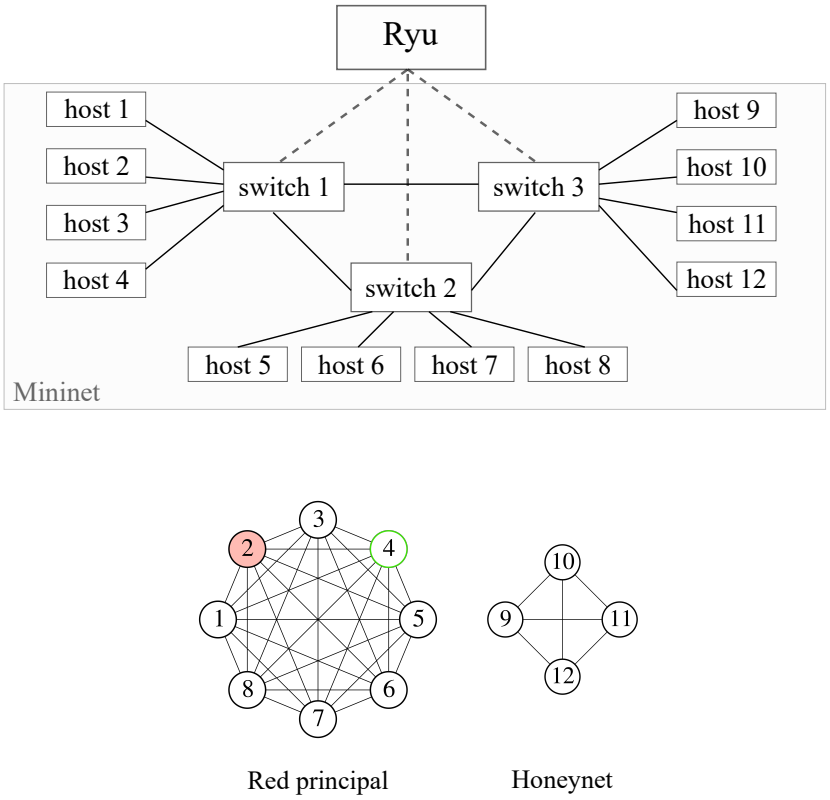


Figura 4.1: Topología de la red (arriba) y red lógica (abajo) del primer entorno estudiado.

La parte de arriba de la Figura 4.1 muestra la topología real de la red: las líneas sólidas representan conexiones Ethernet y las discontinuas son las conexiones entre el controlador y los switches que se utilizan para intercambiar mensajes de OpenFlow. La parte de abajo de dicha figura muestra la estructura lógica de la red, es decir, los nodos con los que puede intercambiar tráfico cada uno de los hosts. De nuevo, el nodo rojo es el que está inicialmente atacado y el del borde verde es la bandera.

El funcionamiento de las contramedidas en este entorno es el siguiente:

- La bandera puede migrarse a nodos no atacados de la red principal, pero no puede establecerse en ninguna máquina de la honeynet.
- Utilizando los flujos adecuados, cualquier nodo de la red principal puede moverse lógicamente a la honeynet.
- Puede aislarse cualquier nodo de la red.

Seguidamente se ha estudiado el caso de una red corporativa que provee varios servicios desde unas máquinas de la red. Generalmente, dichos hosts suelen mantenerse separados del resto de la red corporativa en lo que se conoce como una *zona desmilitarizada* (DMZ, de sus siglas en inglés de *demilitarized zone*): una red física o una subred lógica que se sitúa entre la red interna de la organización y otra red externa (como Internet). Las conexiones entre las máquinas de la DMZ y las de la red interna suelen estar bloqueadas o muy controladas, permitiendo acceder a ellas a los usuarios registrados. De esta manera, la organización sitúa sus servicios públicos en la DMZ, evitando que una posible intrusión cause daños en la red corporativa [61].

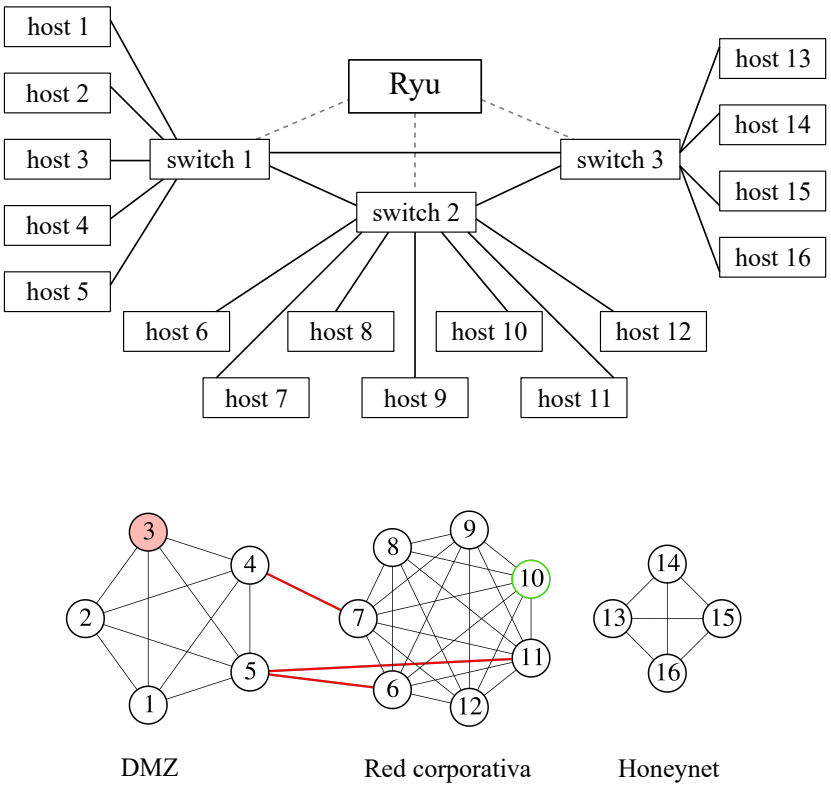


Figura 4.2: Topología de la red (arriba) y red lógica (abajo) del segundo entorno estudiado.

En la red utilizada como demostración representada en la Figura 4.2, se plantea una DMZ compuesta por 5 hosts, una subred corporativa de 7 máquinas y una honeynet de 4 honeypots, formando una red de 16 nodos. Aunque, idealmente, un intruso no podría acceder a la red principal desde la DMZ, en este caso hay tres conexiones que permitirían al intruso tomar el control de un host de la red principal (los arcos en rojo del grafo). Con estas conexiones se quiere

simular el caso en el que el atacante trata de explotar una vulnerabilidad de la zona desmilitarizada (o un error en la configuración de los sistemas de seguridad) para entrar en la red corporativa.

A las condiciones de las acciones establecidas para la red simple hay que añadir que las máquinas de la zona desmilitarizada no pueden ser la bandera y que los hosts de la DMZ también pueden moverse lógicamente a la honeynet.

Los entornos se han estudiado mediante tres experimentos diferentes: experimento A, B y C. Los dos primeros trabajan en el entorno de la red simple y el último se basa en la red que cuenta con la zona desmilitarizada.

El experimento A busca, de la manera más sencilla, demostrar que el defensor es capaz de aprender estrategias de mitigación mediante el juego autónomo. Para ello, se establece el sistema de recompensas más sencillo (el que no contempla el impacto de los ataques ni el coste de las medidas) y el defensor se evalúa enfrentándolo a un atacante "experto". Los detalles de las condiciones del juego y del comportamiento del intruso se explican en la Sección 4.1.

El experimento B, en cambio, utiliza los resultados de las partidas jugadas entre atacantes y defensores con diferentes cantidades de pasos de entrenamiento para evaluar el desarrollo de ambos. Para ello se ha modificado la implementación de MuZero para que dos agentes con entrenamientos diferentes puedan interactuar entre ellos y jugar partidas entre sí. Las características del método de evaluación se explican en mayor profundidad en la Sección 4.2. La probabilidad de que el defensor detecte una máquina comprometida también cambia con respecto al primer experimento: en el caso A dicha probabilidad es $BS/10$, mientras que ahora se establece en $BS/(10N)$, siendo BS el la Puntuación Base de la vulnerabilidad que presenta la máquina atacada y N el número de nodos de la red. Este cambio se justifica en la Sección 4.1 y, obviamente, favorece al atacante y hace que las partidas sean más igualadas. El sistema de recompensas simple también se ha reemplazado por el más complejo: al atacante se le recompensa con el impacto causado en la red y al defensor se le penaliza con dicho impacto y con el coste de las contramedidas implementadas.

El experimento C es similar al anterior, pero en este caso se simula la intrusión de un atacante en una red corporativa a través de la DMZ, tal y como se ha mencionado anteriormente. Al tratarse de un entorno más complejo, el número máximo de pasos permitidos en una partida se ha establecido en 250 en vez de en 150, evitando así que la mayoría de partidas terminen antes de que alguno de los agentes consiga ganar la partida. La Tabla 4.1 resume las características de cada experimento.

	Experimento A	Experimento B	Experimento C
Entorno	Red simple	Red simple	Red con DMZ
Sistema de recompensas	Simple	Complejo	Complejo
Oponente para evaluación	Experto	MuZero	MuZero
Probabilidad de detección	$BS/10$	$BS/(10N)$	$BS/(10N)$
Número máximo de jugadas	150	150	250

Tabla 4.1: Resumen de las características de los experimentos realizados.

Todos los experimentos se han realizado en una máquina virtual con 20 vCPUs @2.2 GHz, 40 GB de RAM y 2 GPU NVIDIA Tesla M10. Los procesadores de la máquina anfitriona son dos Intel Xeon Silver 4114. Los hiperparámetros de MuZero de cada experimento pueden consultarse en el Apéndice A.

4.1. Experimento A

El objetivo principal de este experimento es demostrar que se puede utilizar el algoritmo MuZero para entrenar agentes que resuelvan un juego estocástico parcialmente observado, aunque ni el propio algoritmo ni la implementación utilizada están diseñados para lidiar con entornos no deterministas. Además, esta primera prueba permite sintonizar o corregir algunas características del juego. Teniendo en cuenta que el objetivo final de este trabajo es obtener un defensor inteligente capaz de mitigar la intrusión en una red SDN, las primeras pruebas realizadas se han centrado en tratar de observar el desarrollo de dicho agente.

La implementación de MuZero utilizada realiza pruebas de forma autónoma según se va entrenando el modelo, en la que uno de los agentes puede enfrentarse al otro agente entrenado o a uno que sigue una estrategia programada. En este

caso, se ha fijado la estrategia del atacante para poder evaluar de manera aislada el comportamiento del defensor. Así, durante las pruebas realizadas, el atacante ha seguido la estrategia determinada por el Algoritmo 2. Como es difícil establecer una estrategia de ataque general, se ha optado por que el atacante haga "trampas". Al atacante se le da toda la información del tablero, incluyendo la posición de la bandera y las vulnerabilidades que presentan las máquinas. De esta manera, primero trata de atacar una máquina con la Explotabilidad más alta y que, además, le permita ganar permisos de ejecución (evitando que el defensor lo aísle en las primeras jugadas) y luego trata de capturar la bandera. Se considera también el caso en el que el atacante no tenga acceso a ninguna máquina explotable, en el que el intruso llevaría a cabo una inspección de la red desde sus máquinas infectadas. Cabe mencionar que esa estrategia solo se utiliza para evaluar el modelo y que no afecta al entrenamiento, puesto que todas las partidas que juega MuZero durante la fase de entrenamiento las realiza contra sí mismo.

Algoritmo 2: Estrategia del atacante omnisciente

```
legal_actions ← get_legal_actions(G_A);
exploitable_machines ← get_exploitable_machines(legal_actions, G_G);
attacked_machines ← get_attacked_machines(G_G);
flag_machine ← get_flag_machine(legal_actions, G_G);
if count(exploitable_machines) ≥ 0 then
    if count(attacked_machines) ≥ 2 then
        if flag_machine ∈ exploitable_machines then
            action ← attack_flag_machine;
        else
            action ← attack_most_exploitable;
        end
    else
        action ← attack_most_exploitable;
    end
else
    action ← exploration_action;
end
return action
```

En cuanto al entorno, el primer escenario considerado es el que se muestra en la Figura 4.1 y las vulnerabilidades que presentan las máquinas de la red pueden consultarse en la Tabla 4.2. La probabilidad de detección de un intruso se ha fijado en $BS/10$, situando al defensor en una posición muy ventajosa. Recordemos que la probabilidad de que una acción ofensiva sea existosa es una décima parte de la Explotabilidad de la vulnerabilidad y que la Puntuación Base es mayor o igual a la suma de la Explotabilidad y del Impacto de la misma. Por lo tanto, en general, la probabilidad de detección de una máquina infectada es superior a la probabilidad de comprometerla.

Nodo	BS	Expl.	Imp.	Scope	Vecinos
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

Tabla 4.2: Vulnerabilidades y conexiones de la red simple. La fila roja representa el nodo atacado y la verde se corresponde a la bandera.

En este primer experimento se han entrenado los agentes por 10^4 pasos para cinco semillas diferentes. Las medias (en azul oscuro) y las desviaciones estándar (coloreadas en azul) de la duración de las partidas de evaluación jugadas por MuZero contra el atacante experto y de las recompensas obtenidas por ambos agentes pueden consultarse en la Figura 4.3. Dicha figura muestra únicamente los resultados de las partidas jugadas durante los primeros 5000 pasos de entrenamiento, puesto que a partir de ese punto las métricas seleccionadas se mantenían constantes.

En lo que a la duración de las partidas respecta, puede verse claramente como a partir de los 4000 pasos de entrenamiento el número de jugadas que necesitan los agentes para resolver el juego se mantiene constante. Las pequeñas fluctua-

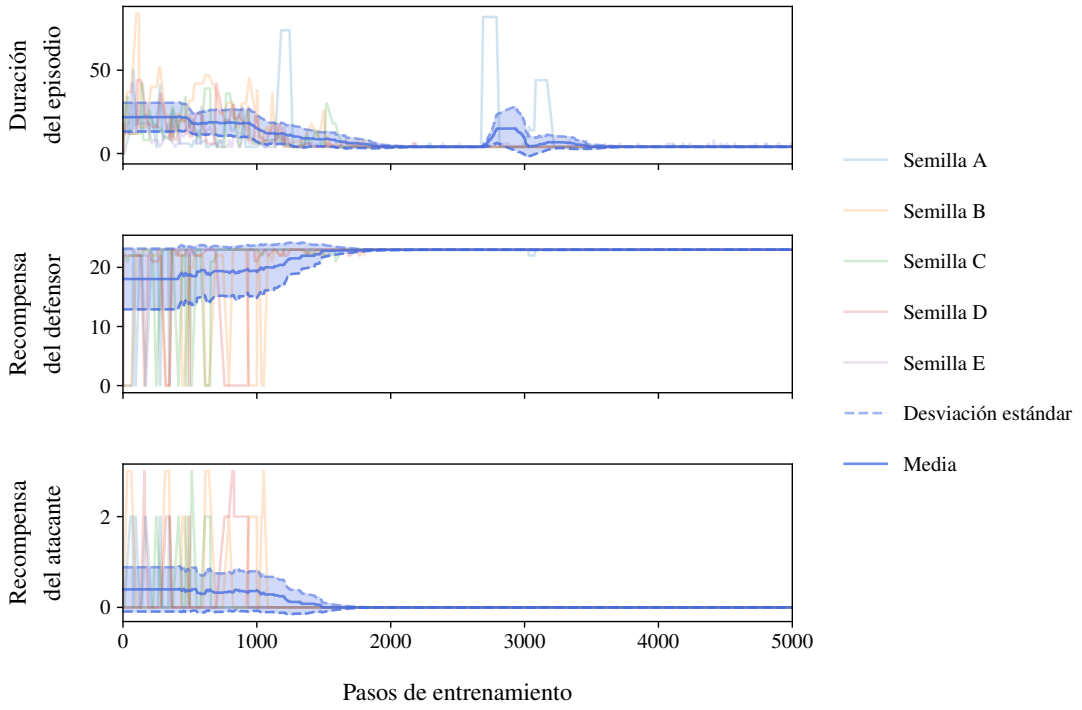


Figura 4.3: Resultados de las partidas de evaluación jugadas entre el atacante experto y el defensor controlado por MuZero durante el entrenamiento.

ciones que aparecen a partir de ese punto se deben a la naturaleza estocástica del juego: a veces los agentes tienen que intentar llevar a cabo una acción varias veces para que esta tenga el efecto deseado.

A la hora de evaluar las puntuaciones obtenidas por los agentes hay que tener en cuenta que (1) la puntuación máxima que puede lograr el defensor es de 23 puntos y que (2) aunque la puntuación máxima del atacante también es esa, como este sigue una estrategia determinada, normalmente obtiene 2 o 3 puntos. Además, como ya se ha mencionado en la Sección 3.2.5, los agentes solo obtienen una recompensa positiva cuando ganan una partida, por lo que la Figura 4.3 también muestra, de forma indirecta, el número de partidas ganadas por cada agente. Dicho esto, puede verse claramente como la media de las recompensas obtenidas por el defensor aumenta a partir de los 1000 pasos de entrenamiento, saturándose al llegar a los 2000 pasos. Así mismo, la media de las recompensas obtenidas por el atacante disminuye hasta llegar a 0. Puede deducirse, por lo tanto, que durante los primeros pasos de entrenamiento la estrategia definida por el atacante es útil para llegar a comprometer la bandera, pero que el defensor aprende rápidamente a mitigar dicho ataque.

Recordemos que el atacante solo sigue la estrategia definida cuando se evalúa al defensor y no durante el entrenamiento: a la hora de actualizar el modelo, MuZero modifica las políticas de los agentes basándose en las partidas y simulaciones realizadas contra sí mismo. En consecuencia, el defensor entrenado es capaz de ganar a un atacante experto sin haberse enfrentado nunca a él. La estrategia de mitigación desarrollada por el defensor durante el entrenamiento es tan eficaz que sería capaz de mitigar una intrusión muy certera.

Estos resultados tienen una parte positiva y otra negativa. Por un lado, se demuestra que MuZero es capaz de resolver este juego estocástico y que el defensor aprende estrategias de mitigación efectivas jugando contra sí mismo. Por otro lado, los resultados indican que el juego está claramente desequilibrado a favor del defensor. Esto puede suponer un grave problema a la hora de obtener agentes realmente inteligentes, puesto que es imprescindible que ambos jugadores vayan mejorando de forma simultánea. Si uno de los agentes estuviese en una clara desventaja y nunca ganase ninguna partida (o ganase muy pocas), no tendría registrado ningún episodio en el que obtuviese una recompensa positiva. Esto podría derivar en la necesidad de muchos más pasos de entrenamiento para obtener una inteligencia mínima o incluso en que el agente mostrase un comportamiento aleatorio. Como ambos agentes se entrenan jugando el uno contra el otro, si el desarrollo de uno de ellos es especialmente lento el de su oponente también lo será, puesto que le será muy fácil resolver los problemas planteados por el adversario.

Por lo tanto, en experimentos posteriores se ha reducido la probabilidad de detección de una máquina comprometida a $BS/(10N)$, siendo BS la Puntuación Base de la vulnerabilidad del host atacado y N el número de nodos de la red. Esta decisión permite que dicha probabilidad se adapte a la dificultad de la red: cuanto mayor sea la red más le costará al atacante encontrar la bandera y más le costará al defensor detectar al atacante. Probablemente existan alternativas más adecuadas o que se ajusten mejor a la realidad, pero el estudio del efecto de la probabilidad de detección en los agentes obtenidos se deja para futuros trabajos.

4.2. Experimento B

En un segundo experimento se ha mantenido la misma red, pero se ha utilizado el sistema de recompensas que sí tiene en cuenta el coste de las contramedidas y el impacto causado por la intrusión (Ecuación 3.8). Para medir el desarrollo de los agentes se han realizado cinco series de entrenamientos con semillas diferentes. En cada serie, se han guardado los modelos obtenidos tras 10^3, 10^4 y 10^5 pasos de entrenamiento. Como MuZero entrena a ambos agentes a la vez, para cada semilla se han obtenido tres atacantes y tres defensores de diferentes niveles. A continuación, se han jugado 100 partidas para cada una de las nueve combinaciones posibles entre los defensores y los atacantes de diferentes niveles. Por lo tanto, se han jugado 900 partidas para cada semilla, 4500 en total. Utilizando los recursos disponibles, se necesitan más de 7 horas para entrenar cada uno de los modelos por 10^5 pasos.

Para cada serie de 100 partidas, se han calculado las medias y las medianas de las recompensas obtenidas por cada agente y de la duración de las partidas ganadas por cada uno de ellos. También se ha registrado el porcentaje de victorias de cada agente. Con el objetivo de facilitar el análisis y mostrar los resultados más representativos, en este apartado se presentan los resultados de las partidas jugadas entre un agente poco entrenado (con 10^3 pasos de entrenamiento) y adversarios de diferentes niveles (con 10^3, 10^4 y 10^5 pasos de entrenamiento). Las tablas completas que muestran todos los resultados obtenidos se presentan en el Apéndice B como resultados complementarios.

La Tabla 4.3 muestra los resultados de las partidas jugadas entre el defensor con 10^3 pasos de entrenamiento y los atacantes de diferentes niveles. Antes de valorar el desarrollo del atacante, merece la pena mencionar que el juego sigue siendo favorable para el defensor: el atacante no gana ninguna de las partidas contra un defensor de su mismo nivel, mientras que el defensor puede ganar la mayoría de ellas (excepto en la semilla E).

En líneas generales, aún partiendo de una situación inicial desfavorable, un atacante entrenado por una cantidad mayor de pasos tiende a ganar más partidas

	Pasos de entrenamiento del atacante	Recompensa				Duración del episodio				PV (%)	
		Atacante		Defensor		Atacante		Defensor			
		Media	Mediana	Media	Mediana	Media	Mediana	Media	Mediana	A	D
Semilla A	10^3	-	-	0,00	0,0	-	-	119,00	119,0	0	1
	10^4	72,78	64,5	0,00	0,0	63,44	46,0	113,00	113,0	18	1
	10^5	123,00	135,0	0,00	0,0	51,27	52,0	72,00	56,0	91	4
Semilla B	10^3	-	-	34,26	0,0	-	-	69,71	69,0	0	82
	10^4	50,88	34,0	54,39	48,0	76,38	69,0	51,63	41,0	16	54
	10^5	100,74	95,0	49,00	0,0	41,23	36,0	58,60	49,0	91	5
Semilla C	10^3	-	-	31,38	0,0	-	-	73,32	70,0	0	68
	10^4	25,00	25,0	39,54	17,0	84,00	84,0	64,30	59,0	1	57
	10^5	101,83	98,0	64,33	85,0	34,02	34,0	41,67	27,0	91	9
Semilla D	10^3	-	-	35,45	15,0	-	-	67,72	59,0	0	69
	10^4	85,50	83,0	55,50	49,0	67,95	62,0	55,85	41,0	42	26
	10^5	114,88	113,5	58,50	50,5	47,33	47,0	36,50	28,0	90	4
Semilla E	10^3	-	-	-	-	-	-	-	-	0	0
	10^4	-	-	-	-	-	-	-	-	0	0
	10^5	132,68	151,0	-	-	47,90	48,0	-	-	97	0

Tabla 4.3: Resultados de las partidas jugadas entre defensores con 10^3 pasos de entrenamiento y diferentes atacantes.

que los agentes menos entrenados. Además, se ve claramente como las medias y las medianas de las recompensas del atacante crecen, por lo que el intruso no solo es capaz de llegar a la bandera, si no que también compromete el máximo de máquinas posible. Similarmente, las recompensas obtenidas por el defensor tienden a disminuir, lo que indica que el defensor necesita más recursos para mitigar la intrusión. La semilla B muestra un caso en el que las recompensas del defensor son mayores al enfrentarse al atacante con 10^4 pasos de entrenamiento que al enfrentarse al de 10^3 pasos. Eso puede deberse a que la estrategia llevada a cabo por el atacante más entrenado sea más agresiva y, en consecuencia, más fácil de detectar (tal vez infecta menos máquinas tratando de llegar rápidamente a la bandera, lo que facilita que el defensor bloquee todos los hosts infectados). La duración de las partidas ganadas por el atacante también disminuye según se va entrenando, por lo que se puede concluir que la estrategia del atacante se vuelve más eficaz y certera. Por otro lado, es interesante ver que la cantidad de partidas ganadas por el atacante más entrenado es similar con todas las semillas, por lo que se puede deducir que las estrategias obtenidas podrían ser bastante similares.

Por otro lado, la Tabla 4.4 muestra los resultados de las partidas jugadas entre el intruso con 10^3 pasos de entrenamiento y los diferentes defensores. Los resultados son incluso más claros que los de la tabla anterior: el rendimiento del defensor aumenta según se va entrenando, lo que deriva en recompensas mayores, en una mitigación más rápida del ataque y en un porcentaje mayor de partidas ganadas. Es interesante el caso de la semilla D, pues incluso manteniendo el número de partidas ganadas (los defensores con 10^4 y 10^5 pasos ganan todas las partidas) las recompensas obtenidas por el defensor aumentan mientras que la duración de las partidas se reduce. Aún así, deberían realizarse más pruebas para determinar si esa diferencia es estadísticamente significativa o no.

		Recompensa				Duración del episodio					
	Pasos de entrenamiento del defensor	Atacante		Defensor		Atacante		Defensor		PV (%)	
		Media	Mediana	Media	Mediana	Media	Mediana	Media	Mediana	A	D
Semilla A	10^3	-	-	0,00	0,0	-	-	119,00	119,0	0	1
	10^4	-	-	20,28	0,0	-	-	84,56	88,0	0	18
	10^5	-	-	127,44	133,5	-	-	25,08	17,0	0	100
Semilla B	10^3	-	-	34,26	0,0	-	-	69,71	69,0	0	82
	10^4	-	-	73,84	81,0	-	-	41,76	33,0	0	97
	10^5	-	-	100,60	115,0	-	-	30,76	25,0	0	100
Semilla C	10^3	-	-	31,38	0,0	-	-	73,32	70,0	0	68
	10^4	-	-	65,55	80,0	-	-	49,37	33,0	0	87
	10^5	-	-	98,15	110,5	-	-	29,06	22,0	0	100
Semilla D	10^3	-	-	35,45	15,0	-	-	67,72	59,0	0	69
	10^4	-	-	130,64	134,0	-	-	27,40	21,0	0	100
	10^5	-	-	131,12	133,0	-	-	24,92	20,0	0	100
Semilla E	10^3	-	-	-	-	-	-	-	-	0	0
	10^4	-	-	131,24	136,0	-	-	26,43	17,0	0	98
	10^5	-	-	131,24	136,0	-	-	26,32	17,0	0	100

Tabla 4.4: Resultados de las partidas jugadas entre atacantes con 10^3 pasos de entrenamiento y diferentes defensores.

Por lo tanto, de este segundo experimento se puede concluir que el juego diseñado permite entrenar agentes que son capaces de cumplir sus objetivos. Respecto al defensor, el agente entrenado mitiga la intrusión maximizando la recompensa obtenida y, en consecuencia, minimizando los recursos utilizados y el daño recibido.

4.3. Experimento C

Como ya se ha mencionado, el último experimento se basa en estudiar el desempeño de los agentes en una red más compleja definida mediante la Tabla 4.5. Comparado con el entorno anterior, este caso de estudio presenta serias dificultades para el intruso: la máquina inicialmente atacada no puede acceder directamente a la bandera, por lo que el atacante debe comprometer varias máquinas para abrirse paso hasta ella. Por otro lado, el número de nodos también es mayor, dificultando la detección de las máquinas atacadas.

Nodo	BS	Imp.	Expl.	Scope	Vecinos
1	7,6	2,8	4,7	Unchanged	2, 3, 4, 5
2	8,3	2,8	5,5	Unchanged	1, 3, 4, 5
3	8,3	2,8	5,5	Changed	1, 2, 4, 5
4	8,8	2,8	5,3	Changed	1, 2, 3, 5, 7
5	9,6	2,8	6,0	Changed	1, 2, 3, 4, 6
6	7,6	4,7	2,8	Changed	1, 2, 3, 4, 5, 7, 8
7	7,2	2,7	3,9	Changed	4, 6, 8, 9, 10, 11, 12
8	10,0	3,9	6,0	Changed	6, 7, 9, 10, 11, 12
9	8,3	2,8	5,5	Unchanged	6, 7, 8, 10, 11, 12
10	7,6	2,8	4,7	Unchanged	6, 7, 8, 9, 11, 12
11	7,2	3,9	2,7	Changed	5, 6, 7, 8, 9, 10, 12
12	7,9	2,5	4,7	Changed	6, 7, 8, 9, 10, 11
13	7,9	2,5	4,7	Changed	14, 15, 16
14	6,8	2,8	4,1	Unchanged	13, 15, 16
15	7,9	2,5	4,7	Changed	13, 14, 16
16	8,8	2,8	5,3	Changed	13, 14, 15

Tabla 4.5: Vulnerabilidades y conexiones de la red con la zona desmilitarizada. La fila roja representa el nodo atacado y la verde se corresponde a la bandera.

Los resultados se han obtenido como en el experimento anterior: para cada semilla se han obtenido tres parejas de agentes y se han jugado 100 partidas para las nueve combinaciones posibles entre esos agentes. La tabla completa de los resultados puede consultarse en el Apéndice B.

La Tabla 4.6 muestra los resultados de las partidas entre el defensor entrenador por 10^3 pasos y los diferentes atacantes. La primera conclusión es que, aún siendo un entorno complejo, tanto el atacante como el defensor son capaces de ganar la partida. En general, dado un defensor fijo, entrenar al atacante por una

	Pasos de entrenamiento del atacante	Recompensa				Duración del episodio				PV (%)	
		Atacante		Defensor		Atacante		Defensor			
		Media	Mediana	Media	Mediana	Media	Mediana	Media	Mediana	A	D
Semilla A	10^3	187,31	173,0	28,71	0,0	170,15	164,0	158,29	163,0	13	17
	10^4	184,09	187,0	1,71	0,0	147,22	149,0	169,86	173,0	46	14
	10^5	122,75	109,0	0,00	0,0	120,13	111,0	220,43	219,0	48	7
Semilla B	10^3	-	-	196,85	203,0	-	-	63,20	57,0	0	99
	10^4	123,19	121,5	190,64	237,5	90,00	69,0	45,43	10,0	26	14
	10^5	97,46	67,0	95,00	54,0	70,57	28,0	129,33	158,0	28	6
Semilla C	10^3	180,22	170,0	192,20	240,0	111,64	104,0	46,20	7,0	55	15
	10^4	173,29	156,5	155,30	237,5	101,06	89,0	56,30	13,0	42	20
	10^5	167,77	159,0	140,55	229,5	120,77	124,0	72,55	15,0	57	18
Semilla D	10^3	184,25	181,5	27,66	0,0	191,50	185,0	162,14	173,0	4	35
	10^4	190,40	176,0	31,50	0,0	137,33	143,0	155,00	189,0	48	10
	10^5	150,06	139,5	38,67	0,0	115,46	108,0	172,67	198,0	48	6
Semilla E	10^3	-	-	67,28	50,5	-	-	113,85	103,0	0	82
	10^4	123,00	123,0	71,52	70,5	234,00	234,0	115,16	98,0	1	62
	10^5	108,26	106,0	16,00	0,0	94,87	78,0	175,00	187,0	39	23

Tabla 4.6: Resultados de las partidas jugadas entre defensores con 10^3 pasos de entrenamiento y diferentes atacantes.

cantidad mayor de pasos suele derivar en un mayor ratio de victorias para el atacante y, a su vez, en un descenso del número de partidas ganadas por el defensor (excepto en el caso de la semilla C, que se analizará al final de este apartado). Sin embargo, el desarrollo de las recompensas obtenidas por el atacante no siempre es el esperado: el más entrenado suele obtener, de media, puntuaciones más bajas que el resto de intrusos. Esto puede deberse a factores estocásticos, pero también podría indicar un cambio en la estrategia del atacante hacia episodios más certeros, en los que trata de comprometer menos máquinas y de asegurar la victoria.

El efecto del aumento del número de pasos de entrenamiento del atacante en el rendimiento del defensor sí que se asemeja más a lo observado en el experimento anterior: según se entrena el atacante, las recompensas obtenidas por el defensor disminuyen, el número de veces que mitiga el ataque baja y suele necesitar más movimientos para bloquear la intrusión. Se puede concluir, por lo tanto, que entrenar más al atacante contribuye a que su política sea más efectiva o, al menos, a que el defensor tenga más problemas para evitar que comprometa la bandera.

Por otro lado, la Tabla 4.7 presenta los resultados de las partidas jugadas entre el atacante menos entrenado y los diferentes defensores. En lo que a partidas ganadas respecta, el entrenamiento del defensor hace que, sin duda, aumente el número de veces que es capaz de mitigar el ataque. Además, excepto en el caso de la semilla C, el defensor con 10^5 pasos de entrenamiento gana en casi todas las partidas. Las recompensas del defensor también aumentan según se entrena el agente, mientras que el número de jugadas que necesita para bloquear al intruso disminuyen.

		Recompensa				Duración del episodio					
	Pasos de entrenamiento del defensor	Atacante		Defensor		Atacante		Defensor		PV (%)	
		Media	Mediana	Media	Mediana	Media	Mediana	Media	Mediana	A	D
Semilla A	10^3	187,31	173,0	28,71	0,0	170,15	164,0	158,29	163,0	13	17
	10^4	53,00	53,0	93,22	67,5	194,00	194,0	89,32	81,0	1	76
	10^5	-	-	206,16	225,0	-	-	45,36	29,0	0	100
Semilla B	10^3	-	-	196,85	203,0	-	-	63,20	57,0	0	99
	10^4	-	-	224,11	230,5	-	-	40,78	28,0	0	100
	10^5	-	-	224,11	230,5	-	-	40,78	28,0	0	100
Semilla C	10^3	180,22	170,0	192,20	240,0	111,64	104,0	46,20	7,0	55	15
	10^4	176,12	176,0	153,77	237,0	113,10	105,0	54,77	10,0	42	26
	10^5	149,97	123,0	194,92	239,0	87,43	81,0	38,38	11,0	60	13
Semilla D	10^3	184,25	181,5	27,66	0,0	191,50	185,0	162,14	173,0	4	35
	10^4	-	-	177,70	209,5	-	-	53,30	36,0	0	88
	10^5	-	-	215,01	231,0	-	-	53,30	36,0	0	99
Semilla E	10^3	-	-	67,28	50,5	-	-	113,85	103,0	0	82
	10^4	-	-	137,35	159,0	-	-	72,53	59,0	0	97
	10^5	-	-	223,43	225,5	-	-	37,90	33,0	0	100

Tabla 4.7: Resultados de las partidas jugadas entre atacantes con 10^3 pasos de entrenamiento y diferentes defensores.

Hay un par de casos que merecen analizarse más en profundidad. En la semilla B, por ejemplo, el defensor parte de una situación muy ventajosa (el defensor con 10^3 pasos de entrenamiento gana casi todas las partidas) y ya es capaz de ganar todas las partidas tras 10^4 pasos de entrenamiento. Lo interesante aquí es que no se ve una degradación en la estrategia del defensor, ya que los resultados del agente más entrenado son idénticos.

Un caso similar es el de la semilla D, en el que, aun manteniendo la media y la mediana de las duraciones de las partidas constante, el defensor sí que es capaz de aumentar su recompensa al pasar de 10^4 a 10^5 pasos de entrenamiento.

En lo que a los resultados de la semilla C respecta, es difícil determinar a qué se deben, pero pueden indicar que, al tratarse de un entorno mucho más complejo que el anterior (el aumento de nodos conlleva un aumento de las acciones disponibles), se necesitarían más pasos de entrenamiento para asegurar que los agentes llegan a desarrollar una estrategia funcional.

En general, estos experimentos demuestran que el juego puede extenderse para representar redes más complejas, aunque es difícil prever el número de pasos necesarios para que los agentes aprendan a resolver el entorno.

Capítulo 5

Conclusiones y trabajo futuro

En este trabajo se ha presentado un sistema inteligente entrenado mediante MuZero capaz de mitigar intrusiones en redes SDN de forma autónoma. La interacción entre el atacante y el defensor se ha planteado como un juego de Markov parcialmente observado y las contramedidas se han implementado en una red SDN basada en OpenFlow utilizando el controlador Ryu.

En general, de los experimentos realizados se puede concluir que el juego diseñado para representar la intrusión es funcional y que se pueden entrenar agentes que aprendan a resolverlo utilizando MuZero. Los resultados del experimento A muestran que es realmente importante el diseño del juego a la hora de entrenar a ambos agentes de forma paralela. Además, de cara a una futura implementación, queda claro que con un sistema de detección lo suficientemente efectivo, MuZero sería capaz de mitigar una intrusión de forma precisa. El haber reducido la probabilidad de detección de las máquinas comprometidas, aparte de mejorar el funcionamiento del propio juego, hace que la simulación de la intrusión sea más realista. En realidad, la interacción entre un atacante y un defensor no sería como la de un juego por turnos: el intruso trataría de comprometer las máquinas y el defensor iría tomando las contramedidas pertinentes de forma reactiva según fuese detectando los ataques. Si el juego fuese totalmente determinista, cada jugada cambiaría el estado del juego y todos los episodios terminarían siendo idénticos. Al bajar la probabilidad de que una acción tenga el efecto esperado, hay muchas acciones que no cambian el estado del tablero, por lo que la forma de actuar de los agentes se vuelve más reactiva.

El experimento B muestra que el entrenamiento basado en la interacción entre los dos agentes da lugar a estrategias efectivas: el atacante aprende a comprometer la bandera y, lo que es más importante, el defensor consigue mitigar el ataque. Cabe destacar que ninguno de los agentes conoce las vulnerabilidades de las máquinas. En consecuencia, estrategias como atacar las máquinas más vulnerables o mover la bandera a una máquina con una Explotabilidad muy baja son comportamientos aprendidos por los agentes mediante las partidas jugadas y las simulaciones realizadas durante el entrenamiento.

Finalmente, el experimento C demuestra que el juego diseñado puede utilizarse para estudiar entornos diferentes. Es especialmente remarcable que el atacante es capaz de resolver el juego, teniendo en cuenta que tiene que comprometer varias máquinas para llegar a la bandera. Por otro lado, los resultados sugieren que, ante entornos más complejos, se necesitarían más pasos de entrenamiento o una sintonización más efectiva de MuZero para asegurar que los agentes aprenden.

En lo que a futuros trabajos respecta, cada uno de los bloques presentados (el juego, el controlador y el entorno de simulación) puede mejorarse y desarrollarse de manera independiente. Teniendo en cuenta que el objetivo final de este proyecto es que un sistema inteligente sea capaz de mitigar una intrusión en una red real, hay varias modificaciones que deben llevarse a cabo para llegar a esa meta.

En cuanto al juego, algunas de las acciones deberían modificarse mínimamente para adecuarse al entorno estudiado (como se ha hecho al pasar de la red simple a la que presenta la zona desmilitarizada). También podrían añadirse nuevas contramedidas o sistemas de recompensas diferentes con el fin de cambiar el comportamiento de los agentes. Aún así, es importante destacar que el haber utilizado el estándar CVSS para definir las características de las vulnerabilidades facilita que un sistema parecido al presentado en este trabajo se utilice en un entorno real.

En cuanto a MuZero, el primer paso sería realizar una sintonización a conciencia del algoritmo. Como ya se ha mencionado antes, los hiperparámetros utilizados en este trabajo han resultado funcionales, pero no se han sintonizado de forma

metódica: la mayoría de ellos han sido propuestos por otros investigadores o por el propio autor de la implementación de MuZero utilizada. Hay que tener en cuenta que incluso utilizando los mismos parámetros el rendimiento del entrenamiento varía de un juego a otro, por lo que habría que sintonizar cada versión del juego o buscar un método de sintonización relativamente rápido que permitiese adaptar los parámetros a la red estudiada o a las condiciones del juego. Una vez solucionado el problema de la sintonización, se podría modificar el propio algoritmo para adecuarlo a juegos estocásticos o incluso a juegos en los que participan más de dos agentes. Esto podría mejorar el rendimiento del modelo y permitir simular situaciones más complejas en las que varios atacantes independientes tratan de comprometer las máquinas de la red.

Cabe destacar también que la implementación de MuZero utilizada ha estado en desarrollo durante el transcurso de este trabajo, por lo que en el futuro podrían existir versiones mejoradas de esta implementación que permitiesen realizar entrenamientos computacionalmente más eficientes, facilitando así el diseño de otros juegos o la puesta en marcha de los modelos en entornos reales.

Respecto al controlador, podrían utilizarse flujos más complejos para simular mejor durante la fase de entrenamiento la red en la que se desplegará el sistema. Aún así, las capacidades de Ryu son adecuadas para utilizarlo en un entorno de producción.

El entorno de simulación podría reemplazarse o modificarse en varias fases antes de utilizar el modelo en una red física. La primera modificación, la cual puede realizarse directamente en Mininet, sería especificar más las características de la red: la velocidad de conexión, los puertos abiertos en cada máquina o los servicios disponibles en cada una de ellas pueden establecerse de manera sencilla en ese entorno y ayudaría a emular entornos más realistas. En consecuencia, tal vez habría que modificar el juego para permitir que los nodos presentasen más de una vulnerabilidad y que el atacante pudiese elegir cuál de ellas tratar de explotar. El siguiente paso podría ser simular intrusiones en un entorno de red desplegado en OpenStack, un software libre de código abierto que permite crear infraestructuras de computación en la nube. OpenStack crea una red de máquinas virtuales

conectadas entre ellas mediante switches virtuales que utilizan OpenFlow, por lo que el despliegue en dicha plataforma sería similar al presentado en este trabajo. En ese momento, los resultados obtenidos en esa plataforma serían idénticos a los que se obtendrían en una red compuesta por switches y hosts físicos. El último paso, por lo tanto, sería hacer que el modelo interactuase con el controlador de una red de switches físicos Openflow.

Para ello habría que añadir un sistema de detección de intrusiones y de monitorización del estado de los nodos de la red. De esta manera, una vez detectado el ataque, un modelo entrenado podría servir de guía para mitigar la intrusión e incluso llegar a bloquear al atacante de forma autónoma. Una posible aplicación podría ser la respuesta a ataques de día cero. Actualmente, el departamento de Ciberseguridad Industrial de Ikerlan está trabajando en ello dentro del proyecto Égida. Por lo tanto, la integración con ese trabajo es también uno de los asuntos que habrá que abordar en el futuro.

Apéndice A

Hiperparámetros de MuZero

Este apéndice presenta los hiperparámetros de MuZero. Tal y como puede verse en la Tabla A.1, la configuración ha sido casi idéntica en los tres experimentos realizados, por lo que se han marcado en negrita los parámetros que cambian de un experimento a otro. Los nombres de los parámetros están directamente en inglés con el objetivo de facilitar su identificación en la implementación de MuZero.

Hiperparámetro	Valor		
	Experimento A	Experimento B	Experimento C
Discount	0,997	0,997	0,997
TD steps	**150**	**150**	**250**
Replay buffer size	$\mathbf{5 \cdot 10^4}$	$\mathbf{1 \cdot 10^5}$	$\mathbf{1 \cdot 10^5}$
Batch size	250	250	250
Unroll steps	5	5	5
MuZero Reanalyze	Enabled	Enabled	Enabled
Value loss weight	0,25	0,25	0,25
Network type	fullyconnected	fullyconnected	fullyconnected
Encoding size	256	256	256
# of representation layers	32	32	32
# of dynamics layers	32	32	32
# of reward layers	16	16	16
# of value layers	16	16	16
# of policy layers	16	16	16
Optimizer	Adam [62]	Adam [62]	Adam [62]
Initial learning rate	$\mathbf{3 \cdot 10^{-3}}$	$\mathbf{4 \cdot 10^{-4}}$	$\mathbf{4 \cdot 10^{-4}}$
Learning decay rate	**0,95**	**1**	**1**
Learning decay steps	**250**	**-**	**-**
L2 reg. weight	$1 \cdot 10^{-4}$	$1 \cdot 10^{-4}$	$1 \cdot 10^{-4}$
Simulations	75	75	75
Dirichlet α	0.25	0.25	0.25
Exploration factor	0.3	0.3	0.3
pUCT c_1	1.25	1.25	1.25
pUCT c_2	19652	19652	19652
Temperature	1,0	1,0	1,0

Tabla A.1: Hiperparámetros utilizados en los diferentes experimentos.

Apéndice B

Resultados complementarios

Este apéndice recoge los resultados de todas las partidas jugadas entre atacantes (A) y defensores (D) entrenados por 10^3, 10^4 y 10^5 pasos, completando así los resultados mostrados en las Secciones 4.2 y 4.3. Las Tablas B.1 y B.2 muestra los resultados obtenidos en la red simple y en la red que tiene una zona desmilitarizada, respectivamente. Dichas tablas muestran las medias y las medianas de las recompensas obtenidas por los agentes, de las jugadas necesarias para acabar las partidas y el porcentaje de victoria (PV) de cada jugador.

	Pasos de entrenamiento		Recompensa				Duración del episodio					
			Media		Mediana		Media		Mediana		PV (%)	
	A	D	A	D	A	D	A	D	A	D	A	D
Semilla 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
Semilla 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
Semilla 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
Semilla 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
Semilla 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	-	17,04	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

Tabla B.1: Resultados de las partidas jugadas entre defensores y atacantes con diferentes pasos de entrenamiento en la red simple.

	Pasos de entrenamiento		Recompensa				Duración del episodio				PV (%)	
			Media		Mediana		Media		Mediana			
	A	D	A	D	A	D	A	D	A	D	A	D
Semilla A	10^3	10^3	187,31	28,71	173,0	0,0	170,15	158,29	164,0	163,0	13	17
	10^3	10^4	53,00	93,22	53,0	67,5	194,00	89,32	194,0	81,0	1	76
	10^3	10^5	-	206,16	-	225,0	-	45,36	-	29,0	0	100
	10^4	10^3	184,09	1,71	187,0	0,0	147,22	169,86	149,0	173,0	46	14
	10^4	10^4	156,05	96,05	128,5	52,0	129,94	93,32	132,0	71,0	36	37
	10^4	10^5	186,38	184,31	201,0	188,0	125,63	49,75	120,0	45,0	16	77
	10^5	10^3	122,75	0,00	109,0	0,0	120,13	220,43	111,0	219,0	48	7
	10^5	10^4	126,54	60,77	114,5	0,0	111,13	121,67	104,0	148,0	48	30
	10^5	10^5	169,69	137,38	159,0	155,5	119,48	68,66	106,0	55,0	23	64
Semilla B	10^3	10^3	-	196,85	-	203,0	-	63,20	-	57,0	0	99
	10^3	10^4	-	224,11	-	230,5	-	40,78	-	28,0	0	100
	10^3	10^5	-	224,11	-	230,5	-	40,78	-	28,0	0	100
	10^4	10^3	123,19	190,64	121,5	237,5	90,00	45,43	69,0	10,0	26	14
	10^4	10^4	183,48	106,29	176,0	68,5	121,33	95,42	106,0	74,0	21	52
	10^4	10^5	149,00	218,38	137,0	239,0	115,78	17,62	11,04	116,0	9	39
	10^5	10^3	97,46	95,00	67,0	54,0	70,57	129,33	28,0	158,0	28	6
	10^5	10^4	95,00	101,35	95,0	97,0	42,00	101,79	42,0	97,0	1	71
	10^5	10^5	145,80	197,14	134,0	242,0	32,52	138,24	138,0	5,0	25	21
Semilla C	10^3	10^3	180,22	192,20	170,0	240,0	111,64	46,20	104,0	7,0	55	15
	10^3	10^4	176,12	153,77	176,0	237,0	113,10	54,77	105,0	10,0	42	26
	10^3	10^5	149,97	194,92	123,0	239,0	87,43	38,38	81,0	11,0	60	13
	10^4	10^3	173,29	155,30	156,5	237,5	101,06	56,30	89,0	13,0	42	20
	10^4	10^4	195,30	152,44	201,0	236,5	119,58	60,00	118,0	12,0	53	18
	10^4	10^5	149,53	202,53	120,0	240,5	88,53	27,00	74,0	8,0	53	28
	10^5	10^3	167,77	140,55	159,0	229,5	120,77	72,55	124,0	15,0	57	18
	10^5	10^4	169,39	187,39	173,0	240,5	120,20	40,78	130,0	8,0	51	18
	10^5	10^5	166,35	177,44	159,0	241,0	101,27	45,25	102,0	7,0	63	16
Semilla D	10^3	10^3	184,25	27,66	181,5	0,0	191,50	162,14	185,0	173,0	4	35
	10^3	10^4	-	177,70	-	209,5	-	53,30	-	36,0	0	88
	10^3	10^5	-	215,01	-	231,0	-	53,30	-	36,0	0	99
	10^4	10^3	190,40	31,50	176,0	0,0	137,33	155,00	143,0	189,0	48	10
	10^4	10^4	168,00	150,72	160,5	152,0	91,67	60,93	70,0	47,0	6	60
	10^4	10^5	237,00	178,47	268,0	194,0	148,93	49,79	146,0	33,0	15	66
	10^5	10^3	150,06	38,67	139,5	0,0	115,46	172,67	108,0	198,0	48	6
	10^5	10^4	142,11	73,30	121,5	4,0	91,36	113,47	73,0	132,0	28	30
	10^5	10^5	234,24	170,05	241,5	238,0	137,61	42,37	143,0	13,0	46	38
Semilla E	10^3	10^3	-	67,28	-	50,5	-	113,85	-	103,0	0	82
	10^3	10^4	-	137,35	-	159,0	-	72,53	-	59,0	0	97
	10^3	10^5	-	223,43	-	225,5	-	37,90	-	33,0	0	100
	10^4	10^3	123,00	71,52	123,0	70,5	234,00	115,16	234,0	98,0	1	62
	10^4	10^4	-	144,13	-	162,0	-	71,35	-	55,0	0	97
	10^4	10^5	-	218,89	-	228,0	-	41,50	-	29,0	0	96
	10^5	10^3	108,26	16,00	106,0	0,0	94,87	175,00	78,0	187,0	39	23
	10^5	10^4	95,87	90,83	81,0	0,0	87,22	96,42	68,0	99,0	23	48
	10^5	10^5	98,94	188,92	81,0	238,0	71,44	36,15	56,0	11,0	18	26

Tabla B.2: Resultados de las partidas jugadas entre defensores y atacantes con diferentes pasos de entrenamiento en la red que tiene una zona desmilitarizada.

Bibliografía

[1] S. Morgan, "2021 report: Cyberwarfare in the C-Suite," *Cybersecurity Ventures*, 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, 2019.

[3] M. Sainz, I. Garitano, M. Iturbe, and U. Zurutuza, "Deep packet inspection for intelligent intrusion detection in software-defined industrial networks: A proof of concept," *Logic Journal of the IGPL*, vol. 28, no. 4, pp. 461–472, 12 2019. [Online]. Available: https://doi.org/10.1093/jigpal/jzz060

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

[5] A. Santos 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 *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, 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 transactions on dependable and secure computing*, vol. 10, no. 4, pp. 198–211, 2013.

[7] K. Hammar and R. Stadler, "Finding Effective Security Strategies through Reinforcement Learning and Self-Play," in *2020 16th International Conference on Network and Service Management (CNSM)*. IEEE, 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 *et al.*, "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*. 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 Communications surveys & tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[11] C. Trois, M. D. Del Fabro, L. C. de Bona, and M. Martinello, "A survey on SDN programming languages: Toward a taxonomy," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 4, pp. 2687–2712, 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," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, p. 69–74, Mar. 2008. [Online]. Available: https://doi.org/10.1145/1355734.1355746

[13] ONF, "OpenFlow switch specification," Open Networking Foundation, Tech. Rep., October 2013. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf

[14] B. Isyaku, M. S. Mohd Zahid, M. Bte Kamat, K. Abu Bakar, and F. A. Ghaleb, "Software Defined Networking Flow Table Management of OpenFlow Switches Performance and Security Challenges: A Survey," *Future Internet*, vol. 12, no. 9, p. 147, 2020.

[15] S. Ahmad and A. H. Mir, "Scalability, Consistency, Reliability and Security in SDN Controllers: A Survey of Diverse SDN Controllers," *Journal of Network and Systems Management*, vol. 29, no. 1, pp. 1–59, 2021.

[16] L. Zhu, M. Monjurul Karim, K. Sharif, F. Li, X. Du, and M. Guizani, "SDN Controllers: Benchmarking & Performance Evaluation," *arXiv e-prints*, p. arXiv:1902.04491, 2019.

[17] N. Z. Bawany, J. A. Shamsi, and K. Salah, "DDoS attack detection and mitigation using SDN: methods, practices, and solutions," *Arabian Journal for Science and Engineering*, vol. 42, no. 2, pp. 425–441, 2017.

[18] T. Xing, D. Huang, L. Xu, C.-J. Chung, and P. Khatkar, "Snortflow: A openflow-based intrusion prevention system in cloud environment," in *2013 second GENI research and educational experiment workshop*. IEEE, 2013, pp. 89–92.

[19] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

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

[21] R. Bellman, "A Markovian decision process," *Journal of mathematics and mechanics*, vol. 6, no. 5, pp. 679–684, 1957.

[22] R. Bellman, "Dynamic programming," *Science*, vol. 153, no. 3731, pp. 34–37, 1966.

[23] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[24] R. B. Myerson, *Game theory*. Harvard university press, 2013.

[25] 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 Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.

[26] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Machine learning proceedings 1994*. Elsevier, 1994, pp. 157–163.

[27] Mitre Corporation, "CVE – Common Vulnerabilities and Exposures." [Online]. Available: https://cve.mitre.org

[28] D. Smallwood and A. Vance, "Intrusion analysis with deep packet inspection: Increasing efficiency of packet based investigations," in *2011 International Conference on Cloud and Service Computing*, 2011, pp. 342–347.

[29] C. Li, Y. Wu, X. Yuan, Z. Sun, W. Wang, X. Li, and L. Gong, "Detection and defense of DDoS attack–based on deep learning in OpenFlow-based SDN," *International Journal of Communication Systems*, vol. 31, no. 5, p. e3497, 2018.

[30] M. Campos and J. Martins, "A SDN-based Flexible System for On-the-Fly Monitoring and Treatment of Security Events," *arXiv e-prints*, pp. arXiv–1806, 2018.

[31] 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, 2020.

[32] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[33] R. J. Williams, *Reinforcement-learning connectionist systems.* College of Computer Science, Northeastern University, 1987.

[34] M. Campbell, A. Hoane, and F. hsiung Hsu, "Deep Blue," *Artificial Intelligence*, vol. 134, no. 1, pp. 57–83, 2002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370201001291

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

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

[37] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. P. Lillicrap, K. Simonyan,

and D. Hassabis, "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm," *CoRR*, vol. abs/1712.01815, 2017. [Online]. Available: http://arxiv.org/abs/1712.01815

[38] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning *et al.*, "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures," in *International Conference on Machine Learning*. PMLR, 2018, pp. 1407–1416.

[39] S. Kapturowski, G. Ostrovski, J. Quan, R. Munos, and W. Dabney, "Recurrent experience replay in distributed reinforcement learning," in *International conference on learning representations*, 2018.

[40] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. Van Hasselt, and D. Silver, "Distributed prioritized experience replay," *arXiv preprint arXiv:1803.00933*, 2018.

[41] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010, pp. 1–6.

[42] K. Morita, I. Yamahata, and V. Linux, "Ryu: Network operating system," in *OpenStack Design Summit & Conference*, 2012.

[43] A. H. Werner Duvaud, "MuZero General: Open Reimplementation of MuZero," 2019. [Online]. Available: https://github.com/werner-duvaud/muzero-general

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

[45] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[46] S. Silvester, A. Tanbakuchi, P. Müller, J. Nunez-Iglesias, M. Harfouche, A. Klein, M. McCormick, OrganicIrradiation, A. Rai, A. Ladegaard, A. Lee, T. D. Smith, G. A. Vaillant, jackwalker64, J. Nises, rreilink, H. van Kemenade, C. Dusold, F. Kohlgrüber, G. Yang, G. Inggs, J. Singleton, M. Schambach, M. Hirsch, M. Komarčević, NiklasRosenstein, P.-C. Hsieh,

Zulko, C. Barnes, and A. Elliott, "imageio/imageio v0.9.0," Jul. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3931847

[47] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring Network Structure, Dynamics, and Function using NetworkX," in *Proceedings of the 7th Python in Science Conference*, G. Varoquaux, T. Vaught, and J. Millman, Eds., Pasadena, CA USA, 2008, pp. 11 – 15.

[48] J. Rapin and O. Teytaud, "Nevergrad - A gradient-free optimization platform," https://GitHub.com/FacebookResearch/Nevergrad, 2018.

[49] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[50] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan *et al.*, "Ray: A distributed framework for emerging AI applications," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 561–577.

[51] M. L. Waskom, "Seaborn: statistical data visualization," *Journal of Open Source Software*, vol. 6, no. 60, p. 3021, 2021. [Online]. Available: https://doi.org/10.21105/joss.03021

[52] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[53] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7: A matlab-like environment for machine learning," in *BigLearn, NIPS Workshop*, 2011.

[54] J. Ren, C. Zhang, and Q. Hao, "A theoretical method to evaluate honeynet potency," *Future Generation Computer Systems*, vol. 116, pp. 76–85, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X20308219

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

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

[57] H. Booth, D. Rike, and G. Witte, "The National Vulnerability Database (NVD): Overview," 2013-12-18 2013. [Online]. Available: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=915172

[58] Forum of Incident Response and Security Teams, Inc, "Common Vulnerability Scoring System version 3.1: Specification Document." [Online]. Available: https://www.first.org/cvss/v3.1/specification-document

[59] Ryu project team, "Switching Hub." [Online]. Available: https://osrg.github.io/ryu-book/en/html/switching_hub.html

[60] 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, 2014.

[61] K. Dadheech, A. Choudhary, and G. Bhatia, "De-militarized zone: A next level to network security," in *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*, 2018, pp. 595–600.

[62] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.