UNIVERSIDAD COMPLUTENSE DE MADRID
FACULTAD DE CIENCIAS FÍSICAS

DEPARTAMENTO DE ACYA

TRABAJO DE FIN DE GRADO

Código de TFG: DACYA-16

Evolución gramatical antagónica para la ciberseguridad

Adversarial grammatical evolution for cybersecurity

Supervisores: José Manuel Velasco Cabo, José Ignacio Hidalgo Pérez

Alberto Benaiges Diez

Grado en Ingeniería Electrónica de Comunicaciones

Curso académico 2023-24

Convocatoria Junio

Calificación: 9.0

Resumen:

En ciberseguridad, es vital mejorar la defensa de redes ante crecientes amenazas, lo cual requiere comprender el impacto de tácticas sofisticadas en dispositivos y sistemas interconectados. Microsoft desarrolló CyberBattleSim, una herramienta que simula escenarios empresariales reales para analizar y crear estrategias de defensa, aunque necesita una configuración compleja. El proyecto de fin de grado propone integrar mecanismos para realizar experimentos de forma automatizada y presenta un primer experimento que cambia la forma de generar la estructura de la red de nodos en CyberBattleSim. Este experimento evalúa cómo un modelo antagónico basado en evolución gramatical, que genera redes de nodos mediante reglas de ensamblaje y programación evolutiva, afecta la eficacia del atacante. El modelo producido por coevolución antagónica selecciona redes óptimas para dificultar las acciones maliciosas, destacando la importancia de la topología de la red en seguridad y proponiendo una defensa basada en acciones pasivas. La automatización del entrenamiento sobre conjuntos de datos permite evaluar y analizar resultados más eficientemente, eliminando la necesidad de interacción manual.

Abstract:

In the realm of cybersecurity, it is essential to enhance network defenses against increasing threats, which requires understanding the impact of sophisticated tactics on interconnected devices and systems. Microsoft developed CyberBattleSim, a tool that simulates real enterprise scenarios to analyze and develop defense strategies, although it necessitates complex configuration. The degree final project proposes integrating mechanisms for automated experiments and presents an initial experiment that alters the node network structure in CyberBattleSim. This experiment assesses how an adversarial model based on grammatical evolution, which generates node networks through assembly rules and evolutionary programming, affects the attacker's effectiveness. The coevolutionary approach selects optimal networks to hinder malicious actions, highlighting the importance of network topology in security and proposing a defense based on passive actions. Automating the training on datasets allows for more efficient evaluation and analysis of results, eliminating the need for manual interaction.

Aunque pasen los años, seguiré apoyándome
en todas y todos vosotros para seguir creciendo.

Incondicionalmente, gracias.

Índice de contenido

1. Introducción **1**
1.1. Motivación 1
1.2. Objetivos 2
1.3. Diagrama de Gantt 3

2. Fundamentación y metodología **4**
2.1. Principios teóricos para las redes de nodos 4
2.1.1. Algoritmos competitivos. Relación antagónica 4
2.1.2. Programación genética 4
2.1.3. Evolución gramatical 5
2.1.4. Aprendizaje por refuerzo 6
2.2. Entorno CyberBattleSim de Microsoft 7
2.3. Conjunto de herramientas Gym de Open AI 8
2.4. Componentes del simulador. Agente y red 8
2.4.1. Agente atacante y algoritmo DQN 8
2.4.2. Red de nodos. Modelado y propiedades 9

3. Desarrollo **11**
3.1. Generación de árboles con gramática 12
3.2. Caracterización de los nodos del árbol 15
3.3. Creación de la interfaz (Gym OpenAI) 16
3.4. Experimento 17
3.4.1. Creación de la primera generación aleatoria 19
3.4.2. Entrecruzado de individuos seleccionados 22
3.4.3. Evaluación de las nuevas generaciones 25

4. Resultados: Evaluación de la primera aproximación **27**

5. Conclusiones **33**

Referencias **35**

6. Anexo **37**

Índice de figuras

1. Diagrama de Gantt . . . 3
2. Fenotipo en forma de árbol binario . . . 14
3. Árbol binario sin representar el genotipo al completo . . . 14
4. Acciones llevadas a cabo por el agente . . . 20
5. Representación de lo acontecido en el episodio 3 . . . 20
6. Curva de la media de recompensas obtenidas durante los tres episodios . . . 21
7. Archivo data.txt con información de cada ejecución . . . 21
8. Método de entrecruzado de las cadenas genotipo . . . 22
9. Representación del algoritmo cruzado-mutado . . . 24
10. Archivo new_genotypes.txt con los cien miembros de la nueva generación . . . 24
11. Estructura mínima de red . . . 27
12. Estructura mínima de red. Representación a través de interfaz gráfica . . . 28
13. Comparativa de resultados dividiendo por el número de nodos posible de cada red . . . 30
14. Experimentos explotando relación *recompensa obtenida - recompensa potencial* . . . 31
15. Comparativa ampliando el número de nodos a treinta . . . 31
16. Red de 10 nodos. Recompensa asociada: 8.61667 . . . 32
17. Red de 10 nodos. Recompensa asociada: 24.11667 . . . 32

Índice de tablas

1. Atributos Técnicos del Nodo . . . 9
2. Acciones disponibles en el nodo Linux . . . 17
3. Acciones disponibles en el nodo Windows . . . 18

Índice de código

1. Reglas de producción simbólicas 6
2. Reglas de producción estrictas. Forma Normal (BNF) 6
3. Lista de caracteres del genotipo 12
4. Clase *Node* de librería *Binarytree* 12
5. Construcción del arbol binario aplicando las reglas de producción 13
6. Asignación de etiquetas a cada tipo de nodo 13
7. Asignación de etiquétas únicas para cada nodo 15
8. Construcción del entorno compatible con la interfaz de Gym 16
9. Registro del entorno compatible 17
10. Generador de genotipos de diez caracteres 19
11. Configuración de las condiciones del agente atacante 19
12. Función de mutación para los genotipos 23
13. Configuración del primer experimento 25
14. Genotipo minimalista 27
15. Función para contabilizar los nodos que conforman una red 29
16. Servicios y vulnerabilidades del nodo Linux 37
17. Servicios y vulnerabilidades del nodo Windows 38
18. Vulnerabilidad en el gestor de contraseñas (CrackKeepPass Vulnerabilities) 39
19. Vulnerabilidad en el gestor de contraseñas (CrackKeepPass Vulnerabilities)(II) 40

1. Introducción

1.1. Motivación

En el ámbito de la ciberseguridad, la evaluación y mejora de la defensa de redes es crucial para hacer frente a las crecientes amenazas en el mundo de internet. A medida que las tácticas, técnicas y procedimientos se sofistican, es necesario tener la capacidad de entender y reconocer su efecto en los dispositivos pero también, a un nivel superior, en los sistemas interconectados que estos conforman.

En este contexto, el entorno de simulación CyberBattleSim[1][2] ha surgido como una herramienta para el análisis y desarrollo de estrategias de defensa. Gracias a este marco de trabajo y las herramientas que utiliza, es posible la simulación de escenarios que se asemejan a entornos en el mundo de las redes empresariales, llegando a poder simular situaciones de amenaza muy similares a la realidad. No obstante, gracias al nivel de abstracción del entorno, las tácticas y técnicas no son replicables en infraestructura real.

CyberBattleSim agrupa los componentes necesarios, pero requiere de una configuración exhaustiva y compleja para replicar una situación real. Además, no dispone de mecanismos que permitan la evaluación y el entrenamiento de modelos basados en conjuntos de datos que requieren de la automatización de experimentos.

El presente trabajo de fin de grado propone la integración de mecanismos para realizar experimentos automatizados y sienta como base un primer experimento que supone una modificación significativa en la forma en que se estructura la red de nodos en CyberBattleSim.

Durante la realización de este proyecto, se tratará de evaluar cómo afecta, sobre la eficacia del atacante a la hora de descubrir, recorrer y conquistar la red, la incorporación de un modelo antagónico basado en evolución gramatical. Este modelo estará enfocado en construir la red de nodos que utiliza el simulador de escenarios para ciberseguridad, CyberBattleSim.

El proyecto propone un cambio en la forma de generar la red de nodos: en lugar de basar la construcción en un algoritmo de construcción aleatorio el proyecto propone la definición de una serie de reglas de ensamblaje entre nodos (gramática) y de un modelo de desarrollo basado en programación evolutiva.

Como punto de partida, se compondrán una serie de individuos (redes aleatorias iniciales) a las que se enfrentará el agente. Siguiendo un modelo producido por coevolución antagónica, se tendrán en cuenta las acciones del atacante para seleccionar las redes óptimas. Se considerarán óptimas las redes que menor puntuación hayan obtenido y se emplearán como padres para engendrar la siguiente generación.

Esta operación se repetirá sucesivamente con el objetivo de encontrar un individuo frente a cuál el agente consiga la menor puntuación posible. De esta manera se trata de encontrar una estructura de red compuesta por dispositivos con una configuración simple, cuya topología dificulte al máximo la capacidad de acción del agente malicioso.

La aproximación que se plantea ayudará a valorar cómo es de relevante la topología de una red en términos de seguridad y expondrá una idea de defensa basada en acciones pasivas. Se pretende realizar todo el estudio en un único experimento del cual se van obteniendo resultados que van marcando el progreso, de forma empírica. Como punto de partida, se simplificará el caso de uso de forma que su sencillez y simpleza sienten correctamente las bases para seguir desarrollando un método que promete una alta escalabilidad.

El proyecto propone una forma de entrenamiento sobre conjuntos de datos que van guiando el camino del experimento según se observan patrones y resultados. Dado que la hipótesis de encontrar una estructura de red compleja difícil de descubrir, recorrer en infectar para el agente atacante, requiere evaluar constantemente el impacto que este tiene sobre la red, tener la capacidad para automatizar esta tarea facilita la evaluación y análisis de los resultados.

En este trabajo se expondrán los pasos a seguir para que sea posible integrar esta herramienta eliminando su parte interactiva para agilizar el proceso de estudio cuando son necesarias muchas ejecuciones en el análisis.

1.2. Objetivos

Los objetivos de este trabajo son:

- Comprender el modo de uso del entorno CyberBattleSim y su aplicación para la simulación de escenarios en ciberseguridad.
- Integrar un método para realizar experimentos con grandes conjuntos de datos que automatice el proceso de análisis y facilite el estudio de los resultados.
- Incorporar como primer experimento base, un modelo antagónico basado en la evolución gramatical, para construir la red de nodos que conforma el entorno.
- Iterar el proceso de generación evolutivo, para crear redes con una alta complejidad de recorrido para el agente atacante.
- Evaluar si la solución propuesta sirve como método para disminuir la capacidad del agente atacante para descubrir, vulnerar e infectar componentes de una red.

Cabe destacar en primer lugar, que el estudio del funcionamiento del simulador, la integración del entorno que representa cada una de las redes y el proceso de automatización de todo el proceso, son pasos primordiales antes de haber planteado el experimento. Además, también ha sido necesario una fundamentación teórica para plantear dicho experimento.

En segundo lugar, a la hora de evaluar los resultados del experimento, el punto de partida del entorno lo representa esa primera generación de individuos cuya topología es aleatoria. Una vez se realiza el tratamiento de la información aplicando el modelo antagónico de evolución gramatical, se visualiza el efecto que este tiene sobre la situación inicial.

1.3. Diagrama de Gantt

El desarrollo del proyecto se ha llevado a cabo siguiendo el orden temporal reflejado en el diagrama representado a continuación.

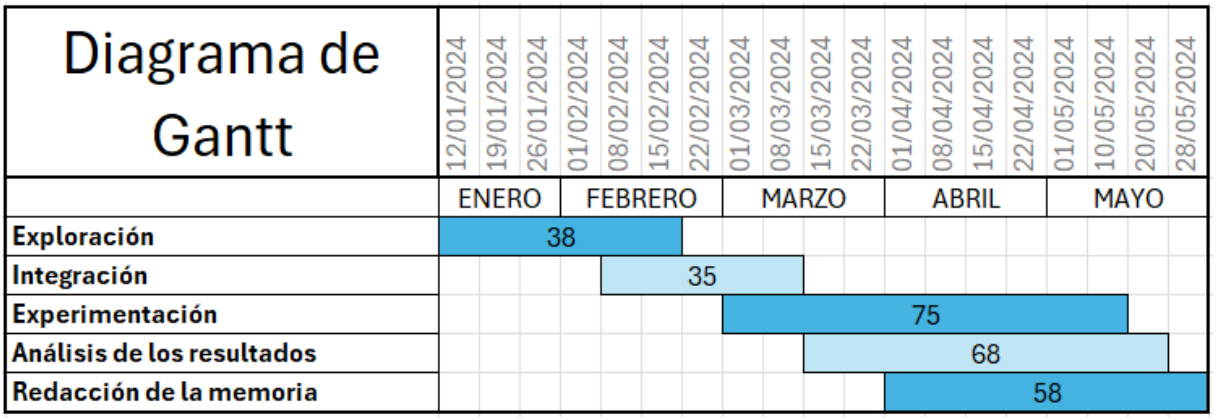

Figura 1: Diagrama de Gantt

Durante la fase de exploración se han investigado diversas fuentes para conocer y comprender el entorno CyberBattleSim, en qué consisten los algoritmos evolutivos y qué valor añaden tanto la programación genética, como la evolución gramatical.

La fase de integración ha consistido en definir una gramática que ayude a interpretar la caracterización en bruto (genotipo) como una topología de red real (fenotipo) e incorporarla como interfaz operable en el entorno.

Una vez incorporado el algoritmo de creación de redes al marco de trabajo, se han programado los algoritmos evolutivos basados en programación genética, con los que se realiza el experimento basado en el método antagónico de evolución gramatical. Se han construido los mecanismos necesarios para obtener la información utilizada para evaluar los resultados.

Habiendo concluido el experimento se ha realizado un proceso de análisis de los resultados con el que se ha establecido un punto de partida desde el que se han propuesto posibles mejoras al caso inicial.

Para finalizar, se ha redactado la memoria, englobando todas las partes del proceso de realización del proyecto.

2. Fundamentación y metodología

En este capítulo se explican los fundamentos que se siguen a la hora de llevar a cabo el desarrollo del experimento. Los individuos a estudiar son redes de nodos y vienen definidos por un genotipo que se forma en base a unas reglas gramaticales. Estas redes/individuos evolucionan según su interacción con el agente y el resultado que este obtiene al tratar de descubrir, recorrer e infectar cada uno de los nodos.

2.1. Principios teóricos para las redes de nodos

2.1.1. Algoritmos competitivos. Relación antagónica

Los algoritmos competitivos se refieren a técnicas y estrategias utilizadas para resolver problemas en entornos donde los recursos son limitados y los participantes compiten entre sí. Esta competencia puede manifestarse de diferentes formas, como maximizar la eficiencia de un algoritmo en términos de tiempo de ejecución, o consumo de recursos, o incluso en términos de optimización de resultados frente a otros algoritmos.

La relación antagónica entre algoritmos competitivos surge cuando dos o más algoritmos compiten directamente en un mismo entorno. En este contexto, cada algoritmo intenta superar a los demás, ya sea ofreciendo resultados más rápidos, soluciones más óptimas o estrategias más efectivas.

Esta competición puede impulsar la innovación y el desarrollo de nuevos algoritmos más eficientes, ya que cada participante busca constantemente mejorar su desempeño para superar a sus rivales.

En este proyecto, se enfrentarán un algoritmo de toma de decisiones para un agente atacante y un algoritmo de construcción de una red de nodos basado programación genética y siguiendo unas reglas de ensamblaje (gramática).

Como referencia para mejorar la red, se hará uso de la realimentación que devuelva el agente. Este método de evolución conjunta en el que interviene la acción de un actor externo, se conoce como coevolución, y es clave para el objetivo final de conseguir una estructura compleja para la red.[3]

2.1.2. Programación genética

La programación genética (GP, Genetic Programming) define una metodología basada en algoritmos genéticos que toma en cuenta principios de la evolución biológica para realizar el desarrollo de programas de una forma automática. De esta forma se busca la obtención de la solución a un problema determinado.

Dentro de una población, cada miembro categorizado con una serie de parámetros, se denomina individuo. Cada individuo posee una composición genética, heredando de la generación anterior un conjunto de genes. Estos genes determinan las características hereditarias de un organismo y comprenden el genotipo. La forma en la que se presentan

estas características una vez se cumple el proceso de desarrollo se denomina fenotipo y es susceptible de cambiar al interaccionar con el entorno.

En este proyecto, cada programa es un individuo y se evalúa el rendimiento que desempeña frente a una tarea a través de una función de ajuste. Tal y como define la biología, la capacidad de cada individuo frente a la resolución de una tarea se denomina su aptitud (fitness). Los individuos con mejores resultados tendrán asociada una mayor aptitud.

Una de las formas más comunes de representación de un programa desarrollado a partir de GP son los árboles de expresión. Estas estructuras definen grandes espacios de búsqueda, son de longitud variable y son ejecutables. Así, pueden ser interpretadas por diferentes algoritmos con el objetivo de proporcionar una solución a un problema específico.

A diferencia de los algoritmos evolutivos, la GP aporta un enfoque en la estructura de los datos, una adaptación a una tarea específica y una exploración de los espacios de búsqueda mas flexible. Busca la producción de programas cuya ejecución representa una solución de la tarea objetivo.

Los operadores característicos de los algoritmos evolutivos, mutación, cruce y selección se mantienen en el proceso de desarrollo de la GP.[3]

2.1.3. Evolución gramatical

La Evolución Gramatical (GE) es un enfoque dentro de la Programación Genética (GP). Este método es capaz de generar un sistema altamente adaptable y fácil de usar, además de permitir un amplio grado de modularidad en sus diseños. La GE utiliza genomas lineales y desarrolla un proceso que va del genotipo al fenotipo (programa), empleando una gramática para introducir estructuras válidas en el espacio fenotípico. Además el valor añadido que aportan algoritmos conformados a través de la definición de una gramática, es la garantía de la viabilidad de todas las soluciones. En el caso de la programación genética, muchas soluciones no son viables ya que al evaluar el programa que conforman no respetan las restricciones que puedan existir. Sin embargo, al imponer unas reglas de producción, se fuerza a que dicha solución sea viable habiendo configurado la gramática para que se respeten las restricciones del problema.

La gramática proporciona un mecanismo único para usar en la GP, permitiendo describir numerosas estructuras complejas. Evolucionar una estructura directamente puede causar problemas, como el uso de un operador genético no válido (cruce) y, posiblemente más importante, la generación de individuos inviables. Esta es otra ventaja de la GE, ya que la búsqueda se realiza en la gramática (es decir, en la secuencia necesaria para producir la estructura deseada). Este componente convierte a la gramática en una herramienta valiosa dentro de la computación evolutiva.[4]

La forma de expresión más habitual para representar la gramática es a través de reglas de producción. Esta notación se conoce como Forma Normal de Backus-Naur (BNF). Las gramáticas en BNF se componen de símbolos no terminales, que generalmente son ele-

mentos concretos (por ejemplo, +, -, etc.), y de reglas de producción (no terminales)[5][6].

Una gramática en BNF se define mediante una cuádrupla N, T, P, S, donde:

- N: denota el conjunto de símbolos no terminales
- T: conjunto de símbolos terminales
- P: reglas de producción
- S: símbolo inicial (comienzo del proceso)

En este proyecto se establece una gramática básica que identifica la forma de construcción de una red de componentes informáticos. Se definen como nodos no terminales aquellos conexión a otros dispositivos, y como nodos terminales como aquellos desde los que no se puede acceder a otro diferente desde el que se ha llegado.

Los dispositivos no terminales ejecutan un sistema operativo Linux y los terminales un Windows:

```
S = [Linux,<hijoIZQ>,<hijoDRCH>]

<nodo>      := [Linux,<hijoIZQ>,<hijoDRCH>] | [Windows,null,null]

<hijoIZQ>   := <nodo>
<hijoDRCH>  := <nodo>
```

Código 1: Reglas de producción simbólicas

```
S = [N,<expr>,<expr>]

    <expr> := [N,<expr>,<expr>]
            | [T,null,null]
```

Código 2: Reglas de producción estrictas. Forma Normal (BNF)

2.1.4. Aprendizaje por refuerzo

El aprendizaje por refuerzo es un tipo de aprendizaje automático en el que un modelo entrenado aprende a realizar acciones óptimas en un entorno para obtener recompensas máximas. Se compone de tres elementos: los agentes, el entorno y las recompensas.[9]

1. **Agentes:** Son entidades que interactúan con el entorno y buscan alcanzar un objetivo final. Realizan acciones que modifican el estado del entorno y reciben recompensas según el valor de la acción. Los algoritmos de aprendizaje por refuerzo permiten a los agentes realizar acciones basadas en una política que maximiza las recompensas.

2. **Entorno:** Está compuesto por dos partes: el espacio de acción y el espacio de observación. El espacio de acción son las acciones que el agente puede realizar en el entorno, mientras que el espacio de observación es toda la información disponible para el agente sobre el entorno.

3. **Recompensas:** Se otorgan en función de una función de recompensa que indica al agente la corrección de la acción a medida que se acerca o se aleja del objetivo final. Las acciones válidas recompensan al agente con un valor positivo, mientras que las acciones inválidas lo penalizan con un valor negativo.

El aprendizaje por refuerzo es esencialmente un problema de optimización, pero se diferencia de las técnicas de optimización tradicionales porque involucra el concepto de recompensa contra valor. La recompensa es un beneficio instantáneo donde el agente elige la acción que producirá la mayor cantidad de recompensa inmediata, mientras que el valor mira más allá del estado actual para obtener la mayor recompensa a largo plazo.

El proceso implica que el agente realice una acción en su espacio de acción, modifique el estado del entorno, reciba una observación del nuevo estado y una recompensa por la acción realizada, y finalmente seleccione la próxima acción basándose en esta nueva observación. Este ciclo se repite hasta que se alcanza un estado final.

En el caso de este proyecto, el agente interactúa de manera competitiva con el entorno y el rendimiento que obtiene se utiliza para valorar la capacidad de la propia red. Por cada episodio el agente toma las decisiones pertinentes utilizando los medios disponibles en función de la configuración del nodo y el éxito que obtiene, otorgan un valor de recompensa que acumula y se retiene hasta el final de la evaluación. Una vez finaliza su acción, el valor total de la recompensa categoriza la calidad del desempeño del agente. Según obtenga mayor o menor recompensa sobre el diseño de la red, se elegirán o descartaran entornos.

2.2. Entorno CyberBattleSim de Microsoft

CyberBattleSim es una plataforma de investigación desarrollada por Microsoft[1] que permite experimentar con agentes automatizados en una red empresarial simulada de manera abstracta, donde dichos agentes se adiestran mediante algoritmos de aprendizaje por refuerzo para atacar o defender nodos de la red. La eficacia de los agentes se evalúa según el número de pasos que necesitan para cumplir sus objetivos y las recompensas totales que acumulan durante el entrenamiento. El entorno que crea CyberBattleSim es una red de nodos conectados por una serie de protocolos de conexión de red diferentes.

CyberBattleSim utiliza estos entornos para modelar problemas de seguridad de red al introducir un atacante en la red. Este atacante representa una entidad maliciosa que intenta tomar el control de la red. CyberBattleSim tiene una topología de red fija y un conjunto de vulnerabilidades que los agentes atacantes pueden utilizar para moverse lateralmente en la red.

El objetivo del atacante es descubrir, recorrer y controlar, aprovechando las vulnerabilidades, de toda o cierta parte de la red

2.3. Conjunto de herramientas Gym de Open AI

Dentro del marco de trabajo de CyberBattleSim se incorpora el conjunto de herramientas Gym de OpenAI.[10] Gym es una biblioteca de Python de código abierto diseñada para desarrollar y comparar algoritmos de aprendizaje por refuerzo. Ofrece una API estándar que facilita la comunicación entre los algoritmos de aprendizaje y los entornos, y proporciona un conjunto de entornos que cumplen con dicha API.

En el caso de este proyecto, no se utilizará un entorno ya definido. En su lugar, se replicará la estructura y el uso de las herramientas de Gym, para añadir una interfaz nueva que siga siendo compatible pero que cumpla con los cambios que se pretenden introducir.

2.4. Componentes del simulador. Agente y red

2.4.1. Agente atacante y algoritmo DQN

El funcionamiento del agente atacante, tal y como está configurado de base en el marco de trabajo, se basa en aprendizaje por refuerzo. Explota vulnerabilidades de movimiento lateral definidas en la matriz MITRE ATT&CK, y emplea un algoritmo denominado DQN[11] para tomar decisiones dentro del entorno simulado. En el aprendizaje por refuerzo, el agente interactúa con el entorno, seleccionando acciones en función del estado actual y recibiendo recompensas por esas acciones. El objetivo es maximizar la recompensa total obtenida a lo largo de una serie de interacciones.[9][12]

La base del aprendizaje por refuerzo radica en el concepto de la función de valor, que estima la recompensa acumulada esperada a partir de un estado dado. En el contexto de DQN, la función Q estima la calidad de tomar una determinada acción en un estado concreto. Durante el proceso de entrenamiento, el algoritmo DQN actualiza iterativamente la función Q a medida que el agente interactúa con el entorno.

En lo que compete a este proyecto, las funciones que puede realizar el agente desde el nodo en el que se encuentra (estado actual) son las siguientes:

1. **run_attack(Node_ID, LOCAL-Vulnerability)**: Ejecutar un ataque e intentar explotar una vulnerabilidad en el nodo especificado.
2. **run_remote_attack(Node_ID, target_Node_ID, REMOTE-Vulnerability)**: Ejecutar un ataque remoto desde el nodo especificado para explotar una vulnerabilidad remota en el nodo objetivo especificado.
3. **connect_and_infect(Node_ID, target_Node_ID, service_name, credentials)**: Instalar el agente en una máquina remota utilizando las credenciales proporcionadas a través del servicio asociado.

Cuando las acciones son exitosas, se descubre, vulnera o infecta un nodo, implica un cambio de estado del entorno, y van ligadas a una recompensa acumulativa.

2.4.2. Red de nodos. Modelado y propiedades

La red se construye en base a un grafo que conforma una estructura de nodos que contienen diferentes propiedades. La posición de los nodos y por tanto la estructura del grafo se decide de forma aleatoria.

El componente de modelado de red es donde el usuario puede crear y ajustar la abstracción de la topología de red. Esta topología de red se visualiza a través de un grafo. Los nodos representan un ordenador o sistema informático en la red empresarial. Los bordes (*edge*) dirigidos señalan hacia otro nodo, obtenido mediante la explotación de una vulnerabilidad o la conexión a través de una credencial filtrada.[14]

Además de agregar o eliminar nodos del grafo, un usuario puede editar varios atributos de un nodo, incluyendo: valor intrínseco, vulnerabilidades, servicios, puertos disponibles y firewalls. Estos atributos están definidos dentro del proyecto CyberBattleSim y se describen en la tabla a continuación. Muchos de estos atributos tienen sus propiedades anidadas, lo que permite al usuario especificar finamente el comportamiento de un nodo.

Nombre	**Descripción**
servicios	Lista de puertos/protocolos a los que el nodo está escuchando
vulnerabilidades	Lista de vulnerabilidades conocidas para el nodo
valor	Valor intrínseco del nodo (se traduce en una recompensa si el nodo es comprometido)
propiedades	Propiedades de los nodos, algunas de las cuales pueden implicar más vulnerabilidades
firewall	Configuración del firewall del nodo
agente instalado	¿Se instaló un agente de ataque en el nodo? (es decir, ¿está comprometido el nodo?)
nivel de privilegio	Nivel de escalada
reinstalable	¿Puede el nodo ser reinstalado por un agente defensor?
cadena de propiedad	Cadena mostrada cuando el nodo es comprometido
estado	Estado de la máquina: en ejecución o detenido

Tabla 1: Atributos Técnicos del Nodo

- **Propiedades generales de un nodo**

Las principales propiedades incluyen: el ID del nodo, el valor intrínseco del nodo, el texto mostrado cuando el nodo es comprometido, un booleano que representa si un adversario ya ha capturado el nodo, y etiquetas de propiedad en función del tipo de servicio que se simula. Los nodos recién creados se instancian con identificadores únicos universales (UUID) como su ID. Esto garantiza que no habrá dos nodos con el mismo ID al ser creados.

Además se incluye un valor que indica si el agente está instalado al inicio en ese nodo (agent_insatlled). De esta forma se acuerda un punto de partida desde el cuál el agente comienza el ataque.[14]

- **Vulnerabilidades de un nodo**

Las vulnerabilidades de los nodos se abstraen teniendo en cuenta los siguientes detalles: tipo de resultado, coste de explotación, tasa de éxito de la explotación, tasa de detección y si la vulnerabilidad requiere acceso local o remoto para ser ejecutada. Un ejemplo de una

vulnerabilidad remota podría ser un sitio públicamente alojado que expone credenciales SSH. Por el contrario, una vulnerabilidad local podría ser extraer un token de autenticación de un dispositivo robado o escalar privilegios a administrador desde dentro del nodo.

CyberBattleSim proporciona varias categorías predefinidas de resultados, que incluyen: credenciales filtradas, referencias filtradas a otros nodos informáticos, datos de usuario filtrados y escalada de privilegios en el nodo. Las vulnerabilidades también pueden ser etiquetadas como remotas o locales. Una vez que se ha explotado una vulnerabilidad, el resultado se presenta al adversario junto con la recompensa asociada al valor del nodo.[14]

- **Servicios de un nodo**

Junto con las vulnerabilidades, un nodo también puede tener servicios en ejecución. Los servicios describen procesos que se ejecutan en un puerto expuesto y que pueden configurarse para requerir credenciales para la autenticación. Por ejemplo, un navegador web puede exponer un servicio HTTPS y una herramienta de transferencia de archivos puede exponer un servicio SSH bajo una credencial.[14]

- **Reglas de firewall en un nodo**

Finalmente, un usuario puede agregar reglas de firewall a un nodo. Las reglas de firewall se pueden utilizar para bloquear o permitir ciertos puertos. Estas reglas pueden ser definidas tanto para el tráfico saliente como para el entrante. Los puertos que no estén permitidos explícitamente en la configuración se asumen automáticamente como bloqueados. Dicho esto, bloquear explícitamente un puerto permite al usuario proporcionar una razón para el bloqueo.[14]

Una vez se han configurado todas las propiedades según convenga, se registra como entorno en Gym de OpenAI para que el agente atacante pueda utilizarse.

3. Desarrollo

A lo largo de este proyecto se tratarán de trasladar el modelo producido por coevolución antagónica de generación de individuos aplicándolo en la construcción de redes que actúan como entorno en CyberBattleSim.

Tal y como se ha mencionado anteriormente, este marco de trabajo desarrolla un simulador que conforman un agente atacante y una red de nodos. Incluye la posibilidad de configurar un agente defensor, pero esta opción se descartará para simplificar el caso de uso a implementar.

El objetivo principal será imponer una serie de modificaciones para crear una red cuya topología y configuración interna sea altamente resistente a los ataques, y sea difícil de navegar para un agente adversario. Para lograr esto, se utilizará el concepto de evolución gramatical antagónica, que implica la generación de estructuras complejas y no intuitivas para dificultar la tarea del atacante.

Se propone un cambio en la forma de generar la red de nodos. En lugar de basar la construcción en un algoritmo de naturaleza aleatoria, se tratará de definir una serie de reglas de ensamblaje entre nodos y un modelo de desarrollo basado en programación evolutiva. Aprovechando el modelo de coevolución se compondrán una serie de individuos (redes aleatorias primitivas) a los que se enfrentará el agente. De las puntuaciones obtenidas, se escogerán las que menor puntuación hayan sacado y se tratarán como padres para generar la siguiente generación.

Se llevará a cabo un estudio exhaustivo de cómo aplicar este modelo antagónico de evolución gramatical en la configuración de redes en CyberBattleSim. Se analizará su impacto en la capacidad del atacante para recorrer la red y comprometer sus nodos, a través de experimentos prácticos haciendo uso del entorno de simulación para evaluar la efectividad de las redes generadas bajo este enfoque.

Esta operación se repetirá sucesivamente con el objetivo de encontrar un individuo frente a cuál el agente consiga la menor puntuación posible.

3.1. Generación de árboles con gramática

Todo el código presentado es de creación propia. Sin embargo, es necesario mantener una similitud en forma y nombres de los diccionarios y funciones que utiliza el resto del marco de trabajo para que exista compatibilidad.

Así mismo, como se explicará en el punto 3.4, toda la configuración se registra como perfil para poder ser utilizado como uno de los entornos definidos en el conjunto de herramientas Gym de OpenAI, tal y como funciona CyberBattleSim.

En este proyecto se introduce la capacidad de generar una topología siguiendo ciertas reglas de construcción. Configurando la gramática de nodos terminales y no terminales, se genera un genotipo compuesto por número pares (no terminales, Linux) e impares (terminales, Windows):

```
[2, 2, 1, 1, 2, 1, 2, 2, 1, 1]
```

Código 3: Lista de caracteres del genotipo

La forma de árbol en la que se ha estructurado la información se basa en la utilización de una clase denominada *Node* que garantiza la forma de árbol binario guardando el valor del nodo, una referencia a la izquierda y una referencia a la derecha. Así se garantiza la gramática implícita. De esta forma, si el nodo es no terminal se le asociaran referencias a otro nodo de izquierda a derecha y si es terminal se dejarán vacías.

```
class Node:
    def __init__(
        self,
        value: NodeValue,
        left: Optional["Node"] = None,
        right: Optional["Node"] = None,
    ) -> None:
        self.value = self.val = value
        self.left = left
        self.right = right
```

Código 4: Clase *Node* de librería *Binarytree*

A través de la interpretación de los genes tal que 2 es un nodo no terminal Linux y 1 refiere a un nodo terminal Windows.

```
"""
Builds a binary tree using the given list of numbers.

Args:
    nums (list): A list of numbers representing the nodes of the binary
   tree.

Returns:
    root: The root node of the binary tree.

"""
if not nums:
    return None

root = node(nums[0])

q = [root]

i = 1
while i < len(nums):
    if not q:  # Check if the queue is empty
        break
    curr = q.pop(0)
    if curr.val % 2 == 0:  # Non-Terminal Node
        if i < len(nums):
            curr.left = node(nums[i])
            q.append(curr.left)
            i += 1
        if i < len(nums):
            curr.right = node(nums[i])
            q.append(curr.right)
            i += 1
return root
```

Código 5: Construcción del arbol binario aplicando las reglas de producción

Para la representación se asignan unas etiquetas provisionales:

```
"""
Assigns labels to the nodes of a binary tree based on the value of each
   node.

Returns:
    The modified root node with labels assigned to each node.
"""
if node is None:
    return


```

```
if node.val % 2 == 0:
    node.val = " linux "  # None-Terminal Node
else:
    node.val = " windows " # Terminal Node

self._assign_labels(node.left)
self._assign_labels(node.right)
return node
```

Código 6: Asignación de etiquetas a cada tipo de nodo

Así, se obtiene el fenotipo:

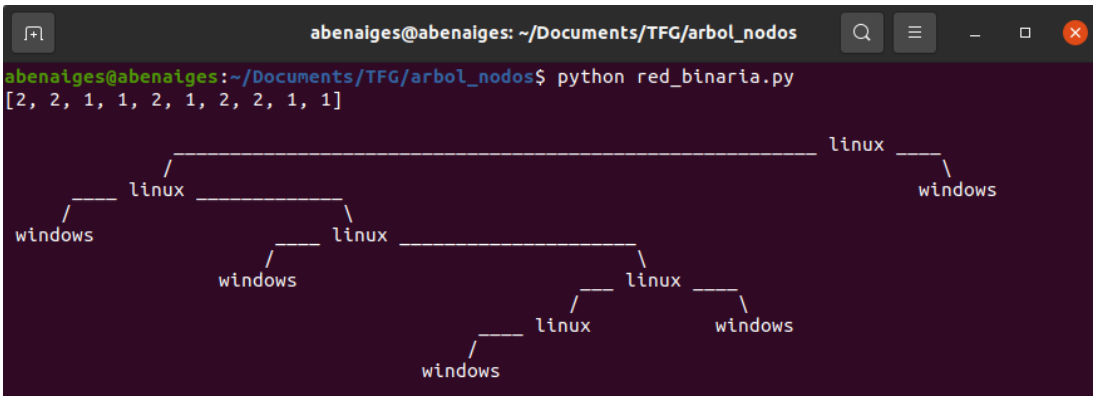

Figura 2: Fenotipo en forma de árbol binario

En el caso presentado, es posible representar todo el genotipo al completo. Como se puede observar, hay una posición vacía en el nodo Linux inferior que no ha llegado a llenarse por falta de miembros.

Puede darse el caso, en el que el fenotipo represente menos miembros de los que contiene el genotipo, debido a que los nodos terminales lo impiden, ocupando todas las posiciones y no generando nuevas.

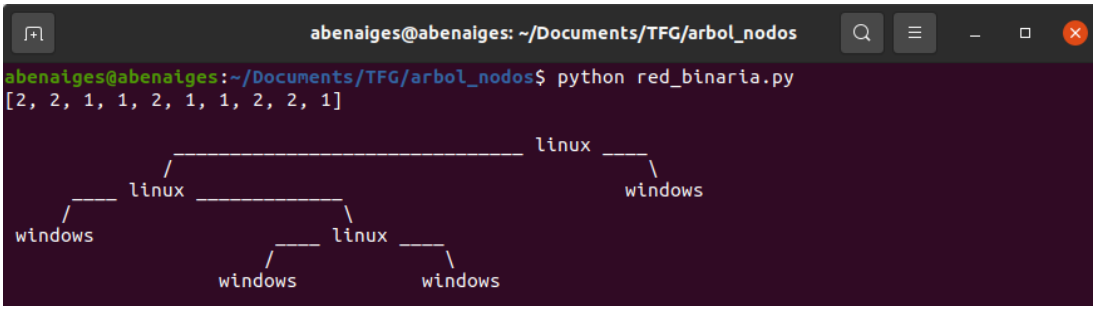

Figura 3: Árbol binario sin representar el genotipo al completo

Como se puede ver en la figura, se representan siete de los diez nodos que conforman el genotipo.

3.2. Caracterización de los nodos del árbol

La característica principal de cada tipo de nodo es su sistema operativo. Esto identifica el nodo y sus propiedades. El marco de trabajo define vulnerabilidades basadas en técnicas de explotación de movimiento lateral que el agente ejecuta de forma local y remota para descubrir e infectar los siguientes nodos.

En primer lugar, es necesario para mantener la compatibilidad con el marco de trabajo, asignar un identificador único a cada nodo. Este identificador permite al agente especificar a qué nodo trata de realizar la explotación de la vulnerabilidad (puede realizar la acción en el nodo actual o en aquel del que tiene la referencia) y además con este identificador, se generan las credenciales de cada servicio. Para un nodo Linux denominado *linux_3* la contraseña ligada a su servicio SSH sería *LinuxPassword!linux_3*. De esta forma la contraseña para utilizar el servicio que consigue la propiedad del nodo, también es única.

```
"""
Given a node, it creates a unique label.
"""
global id_counter

if node.val % 2 == 0:
    node.unique_label = f"linux_{id_counter}"

else:
    node.unique_label = f"windows_{id_counter}"

id_counter += 1
```

Código 7: Asignación de etiquétas únicas para cada nodo

En segundo lugar, se han utilizado las propiedades ya definidas en un ejemplo de interfaz cargado en el marco de trabajo de CyberBattleSim, *chainpattern.py*. Se han mantenido las propiedades, los servicios y la configuración del firewallpreestablecida. Las propiedades del nodo Linux y el nodo Windows se exponen en el anexo y se resumen más adelante en el apartado 3.4.

Por último, para las vulnerabilidades también se ha mantenido parte de la configuración, pero han sido necesarias una serie de modificaciones que garantizan la manera en la que el agente atacante recorre la red. Es clave, al realizar la definición, ser muy cuidadosos con el orden en el que se asocian dichas vulnerabilidades porque es lo que garantizará que se respete la gramática.

En el nodo no-terminal, como se presenta en el código en el anexo (*Servicios y vulnerabilidades del nodo Windows:*6), la vulnerabilidad *ScanBashHistory*, permite al agente conocer cuales son los dos siguientes nodos que penden del actual en el que se encuentra. Emula un escaneo en el historial de comandos del equipo, en búsqueda de referencias a otros dispositivos.

En el código *Vulnerabilidad en el gestor de contraseñas (CrackKeepPass Vulnerabilities) y (CrackKeepPass Vulnerabilities)(II)*6 adjuntado en el anexo, se muestra cómo se han configurado las vulnerabilidades que permiten al agente infectar el siguiente nodo, es decir, avanzar a la siguiente posición.

El agente atacante rompe la seguridad del gestor de contraseñas del nodo local para obtener las credenciales de un servicio de control remoto del nodo que descubrió escaneando el historial de comandos de bash.

Dado que cada nodo no terminal se puede relacionar con otros dos, se identifica qué sistema operativo porta cada nodo a izquierda y derecha, y se les asignan sus propiedades en consecuencia.

3.3. Creación de la interfaz (Gym OpenAI)

Gym, como se expuso en el punto 2.3, es una biblioteca de herramientas de código abierto desarrollada por Open AI que se utiliza para la creación y comparación de algoritmos de aprendizaje por refuerzo (RL). Para incorporar el entorno de estudio de este proyecto ha sido necesario seguir los patrones de configuración que define el estándar de Gym. A través de la generación de un entorno denominado TFGv0, se puede evaluar el rendimiento del agente atacante frente a la red definida.

A continuación se muestra la función que junta todos los pasos necesarios para generar:

1. Un genotipo (Lista de números).
2. Un fenotipo que mantiene la estructura definida por la gramática y asocia las propiedades necesarias a cada nodo (Red de nodos).
3. Un entorno compatible con la interfaz de Gym.

```
class TFGv0(CyberBattleEnv):
    def __init__(self):
        # 1. Create tree
        self.create_tree(nums)

        for node in self.tree.root:
            # 2.1. Assign unique label to each node
            self.assign_unique_label(node)

        for node in self.tree.root:
            # 2.2. Add node information based on node type (Linux or
   Windows)
            self.add_node_info(node)



        # 3. Create network by converting tree nodes into a graph
        self.create_network()

```

```
        # 4. Create environment using the network and vulnerability library
        environment = self.create_environment()

        # Check validation and compatibility with the parent class
        super().__init__(initial_environment=environment)
```

Código 8: Construcción del entorno compatible con la interfaz de Gym

```
def create_environment():
    TFGv0()

    # 5. Register the environment with the gym
    register(
        id="TFG-v0",
        cyberbattle_env_identifiers=chain_sample.ENV_IDENTIFIERS,
        entry_point="tfg.week_20240411.s00_tree_to_cyber:TFGv0",
        kwargs={},
    )
```

Código 9: Registro del entorno compatible

Así queda definida la red de nodos en el formato preciso para poder trabajar con el marco de herramientas de CyberBattleSim. Se puede observar que este es un caso de uso particular, pero la posibilidad de abstracción abre un abanico de posibilidades de configuración para estudio de diferentes topologías de red.

3.4. Experimento

Una vez se ha compuesto la red de nodos y se ha integrado como interfaz ejecutable, se configura la llamada al agente atacante con los parámetros necesarios, siguiendo el modelo de problema propuesto.

A continuación, se deciden los valores de las recompensas ligadas a cada acción del atacante. En la configuración antes expuesta de cada nodo se establecen el coste que supone explotar las vulnerabilidades y apropiarse de los nodos, el número de iteraciones y el número de episodios.

Nodo Linux (Valor: 100)		
Vulnerabilidad	**Descripción**	**Recompensa**
ProbeLinux	Sondeo para verificar si el nodo utiliza Linux	5
ProbeWindows	Sondeo para verificar si el nodo utiliza Windows	0
ScanBashHistory	Escaneo del historial de bash en busca de posibles referencias a otras máquinas	10 & Referencias a los nodos de la derecha e izquierda
ScanExplorerRecentFiles	Escaneo de archivos recientes del explorador de Windows en busca de posibles referencias a otras máquinas	0
SudoAttempt	Intento de hacer sudo como usuario administrador	0
CrackKeepPass	Intento de romper KeepPass y buscar credenciales	50 & Credenciales para conectarse mediante servicios de control remoto (SSH)

Tabla 2: Acciones disponibles en el nodo Linux

Nodo Windows (Valor: 100)		
Vulnerabilidad	**Descripción**	**Recompensa**
ProbeLinux	Sondeo para verificar si el nodo utiliza Linux	0
ProbeWindows	Sondeo para verificar si el nodo utiliza Windows	25
ScanBashHistory	Escaneo del historial de bash en busca de posibles referencias a otras máquinas	0
ScanExplorerRecentFiles	Escaneo de archivos recientes del explorador de Windows en busca de posibles referencias a otras máquinas	0
SudoAttempt	Intento de hacer sudo como usuario administrador	0
CrackKeepPass	Intento de romper KeepPassX y buscar credenciales	50 & Credenciales para conectarse mediante servicios de control remoto (RDP)

Tabla 3: Acciones disponibles en el nodo Windows

Como el proceso pasa por evaluar cuanto se desplaza el agente por la red, se recompensarán en mayor medida las acciones que implican descubrimiento e infección de nodos que no posee el atacante. Además a modo de simplificación, las vulnerabilidades cuya explotación falle (*outcome = m.ExploitFailed (),outcome = m.ProbeFailed ()*) (S*ervicios y vulnerabilidades del nodo Linux:*6) (*Servicios y vulnerabilidades del nodo Windows:*6) no serán recompensadas. Tampoco se configurará un objetivo de conquista, es decir, el atacante finalizará su acción en cada episodio, según acaben las iteraciones.

Por otro lado, se puede observar que los nodos Windows no aportan información sobre la red, dada su condición de nodo terminal. El agente, sí gana recompensa por descubrir estos nodos y ejecutar sus vulnerabilidades asociadas.

Así una vez se conocen tanto la configuración y propiedades de la red como el agente atacante, se procede a plantear el experimento. En primer lugar, se engendra una primera generación de cien redes aleatorias. Se ejecuta el simulador sobre cada una de ellas, obteniendo cien recompensas. Acto seguido se ordenan y se seleccionan las cincuenta peores.

Como se está evaluando el rendimiento de la red frente a la acción del atacante, se consideran las peores recompensas del agente como las óptimas a la hora de reflejar la dificultad que le ha presentado recorrer esas redes.

A continuación, se procede a realizar el entrecruzado y mutado de la redes, siguiendo así con la forma de operar de un algoritmo evolutivo dando forma a la siguiente generación. El entrecruzado consiste en seleccionar aleatoriamente dos candidatos de la lista de cincuenta, dividir sus genotipos a la mitad y unir la primera mitad del primero con al segunda del segundo y viceversa.

Además, con el resultado obtenido del entrecruzado, se realiza una operación de mutación en base a una probabilidad que modifica aleatoriamente un único valor cada genotipo (por cada vuelta), cambiando a par si es impar y viceversa.

3.4.1. Creación de la primera generación aleatoria

Como punto de partida se genera la primera generación de cien genotipos de red, definiendo aleatoriamente diez números pares o impares para cada una.

```
NUMS = [2] + [random.choice([1, 2]) for _ in range(9)]
```

Código 10: Generador de genotipos de diez caracteres

Para cada candidato se crea un entorno que se registra en el marco de trabajo y se evalúa contra el agente. La recompensa obtenida junto con el genotipo se escriben en un archivo.

A continuación se muestra la configuración de la función encargada de incorporar el agente con las condiciones que incluye por defecto en el marco de trabajo. Se han establecido tres episodios de doscientas iteraciones.

```
cyberbattlechain = gym.make(
    "TFG-v0",
)

# Run Deep Q-learning (Attack Agent Configuration)
dqn_learning_run = learner.epsilon_greedy_search(
    # In learner, we set the render parameter to false
    # and we choose by modifiying this list render parameter
    # if print in browser or not
    cyberbattle_gym_env=cyberbattlechain,
    environment_properties=ep,
    learner=dqla.DeepQLearnerPolicy(
        ep=ep,
        gamma=0.015,
        replay_memory_size=10000,
        target_update=10,
        batch_size=512,
        learning_rate=0.01,
    ),
    episode_count=3,
    iteration_count=200,
    epsilon=0.90,
    render=False,
    epsilon_exponential_decay=5000,
    epsilon_minimum=0.10,
    verbosity=Verbosity.Quiet,
    title="DQL",
)
```

Código 11: Configuración de las condiciones del agente atacante

Así, ejecutando el código sobre la red definida en el punto 3.2 del apartado de Desarrollo, se obtiene para cada una de las redes la información referente a cada uno de los episodios.

```
  ## Episode: 3/3 'DQL' ϵ=0.8386, γ=0.015, lr=0.01, replaymemory=10000,
batch=512, target_update=10
Episode 3|Iteration 200|reward:  250.0|last_reward_at:  198|Elapsed Time: 0:00:03|##################################################
  Episode 3 stopped at t=200
  Breakdown [Reward/NoReward (Success rate)]
    explore-local: 3/38 (0.07)
    explore-remote: 10/68 (0.13)
    explore-connect: 0/48 (0.00)
    exploit-local: 2/9 (0.18)
    exploit-remote: 0/1 (0.00)
    exploit-connect: 2/19 (0.10)
  exploit deflected to exploration: 1
              status                    properties                                  local_attacks             remote_attacks
id
linux_0         owned  [MySql, Ubuntu, nginx/1.10.3]  [SudoAttempt, ScanExplorerRecentFiles, ScanBas...  [ProbeLinux, ProbeWindows]
linux_1         owned  [MySql, Ubuntu, nginx/1.10.3]  [SudoAttempt, CrackKeepPassX_left, ScanExplore...  [ProbeLinux, ProbeWindows]
linux_4         owned  [MySql, Ubuntu, nginx/1.10.3]  [SudoAttempt, CrackKeepPassX_left, ScanExplore...  [ProbeLinux, ProbeWindows]
windows_2  discovered                            NaN                                            None  [ProbeLinux, ProbeWindows]
windows_3  discovered                            NaN                                            None  [ProbeLinux, ProbeWindows]
simulation ended
```

Figura 4: Acciones llevadas a cabo por el agente

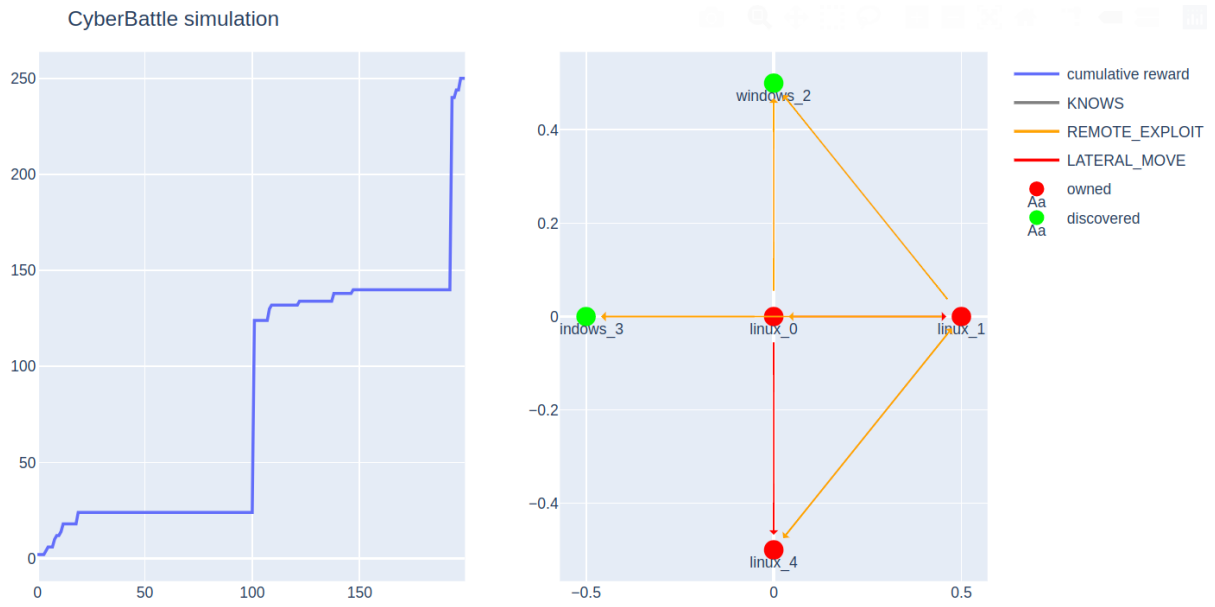

Figura 5: Representación de lo acontecido en el episodio 3

Se obtiene una tabla resumen (Figura 4: 3.4.1) con las acciones llevadas a cabo durante la ejecución del episodio por el agente atacante, una curva que refleja el ritmo de adquisición de recompensas en función de las iteraciones y un gráfico que ejemplifica la situación del escenario de ataque , indicando la posición de los nodos en coordenadas (Figura 5: 3.4.1), al final del episodio.

Además, el último punto de la curva en la figura anterior, representa la recompensa total que el agente a obtenido. Como información adicional, el marco de trabajo incluye la función de representar la media de recompensas obtenidas por cada iteración durante los tres episodios, como se muestra en la figura a continuación.

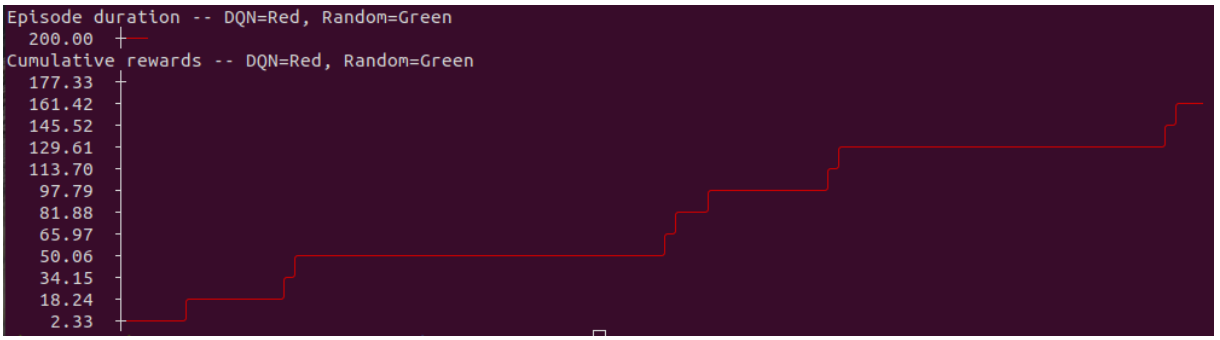

Figura 6: Curva de la media de recompensas obtenidas durante los tres episodios

El valor que se obtiene de realizar la media de recompensas en la última iteración de los tres episodios será el valor de referencia que se asociará al genotipo que define la red candidata. Esta información se volcará en un archivo para su análisis posterior, tal y como se muestra en la siguiente figura.

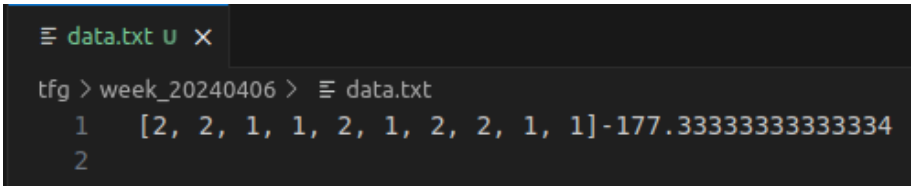

Figura 7: Archivo data.txt con información de cada ejecución

De esta forma se sucederán cien genotipos diferentes que recibirán el mismo tratamiento y se obtendrá una batería de genotipos asociados a la recompensa obtenida tras evaluar su rendimiento en tres episodios, durante doscientas iteraciones cada uno.

3.4.2. Entrecruzado de individuos seleccionados

Para llevar a cabo los mecanismos de entrecruzado y mutación de los candidatos se construye un código que posibilita la capacidad de leer la información volcada en un fichero, tras la realización de las ejecuciones del marco de trabajo. El algoritmo se desarrolla en una serie de pasos:

1. Se formatea la información para poder leer la recompensa obtenida para cada genotipo.
2. Se seleccionan las cincuenta recompensas de menor valor.
3. De los cincuenta genotipos asociados, se escoge aleatoriamente una pareja.
4. Se divide cada miembro por la mitad y se realiza el entrecruzado de la forma que indica la imagen:

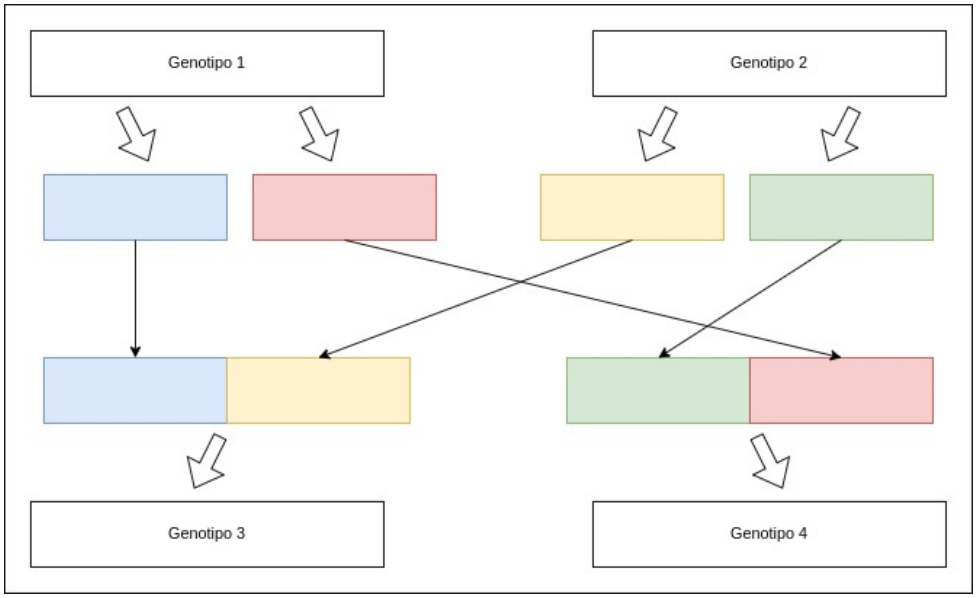

Figura 8: Método de entrecruzado de las cadenas genotipo

5. Se aplica una función de mutación asociada a una probabilidad, para cada uno de los descendientes obtenidos.
6. Se escribe en un archivo los nuevos genotipos.

El proceso de mutación se realizará sobre cada uno de los genes, y se restringe a una única mutación por cada genotipo. Con el propósito de reconocer qué valores mutan, se seleccionan números pares e impares diferentes.

```python
def mutate_genotypes(new_genotype1, new_genotype2):
    # Convert the genotypes to lists of integers
    list_new_genotype1 = [int(num) for num in new_genotype1 if num.isdigit()]
    list_new_genotype2 = [int(num) for num in new_genotype2 if num.isdigit()]

    i = 2

    # Iterate over the first 10 elements of the genotypes
    for i in range(i, 10):

        # With a probability of 0.05, change the value of the genotypes
        if random.random() < 0.05:
            # If the value odd, change it to 6
            if list_new_genotype1[i] % 2 == 1:
                list_new_genotype1[i] = 6
                break  # Exit the loop once a condition is met
            # If the value is even, change it to 5
            elif list_new_genotype1[i] % 2 == 0:
                list_new_genotype1[i] = 5
                break

            # If the value is odd, change it to 6
            if list_new_genotype2[i] % 2 == 1:
                list_new_genotype2[i] = 6
                break
            # If the value is even, change it to 5
            elif list_new_genotype2[i] % 2 == 0:
                list_new_genotype2[i] = 5
                break

    # Return the mutated genotypes
    return list_new_genotype1, list_new_genotype2
```

Código 12: Función de mutación para los genotipos

Así el algoritmo de entrecruzado y mutación al completo se interpretaría conjuntamente tal y como representa la imagen a continuación.

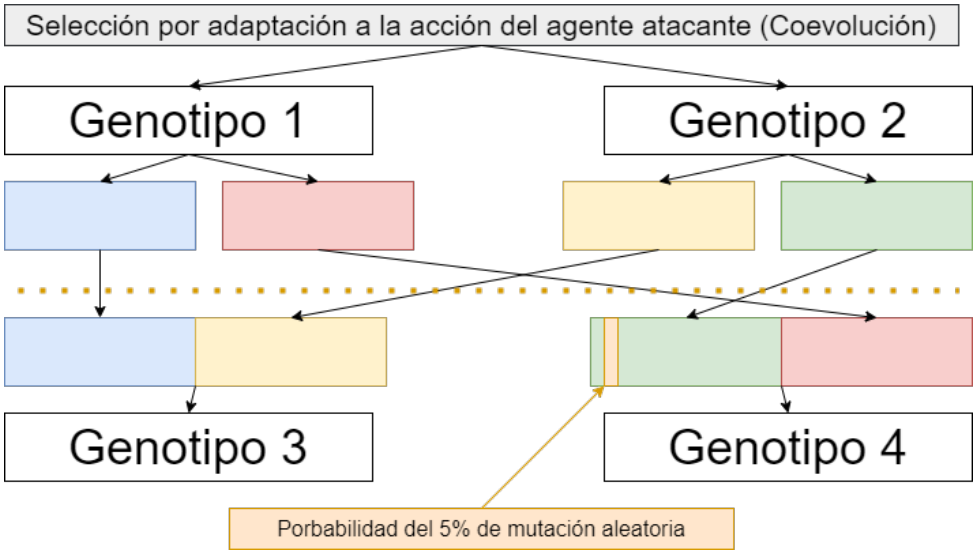

Figura 9: Representación del algoritmo cruzado-mutado

Desde el paso tres, se repetirá el procedimiento para obtener una nueva generación de cien redes candidatas que se volverán a evaluar siguiendo un proceso evolutivo con el fin de generar generaciones que contengan miembros que dificulten al máximo la actividad del atacante.

Así, por cada generación producida tras la evaluación mediante el marco de trabajo CyberBattleSim, se compone un fichero de la forma:

tfg > week_20240406 > new_genotypes.txt

```
1   [2, 1, 1, 2, 2, 1, 1, 2, 1, 1]
2   [2, 2, 1, 2, 2, 2, 2, 2, 1, 1]
3   [2, 1, 1, 1, 2, 1, 1, 2, 5, 2]
4   [2, 2, 1, 1, 2, 1, 2, 1, 2, 1]
5   [2, 2, 2, 2, 6, 2, 1, 1, 2, 1]
6   [2, 1, 1, 1, 6, 1, 1, 1, 1, 1]
7   [2, 1, 1, 1, 1, 2, 2, 2, 2, 2]
8   [2, 1, 1, 1, 2, 2, 1, 2, 2, 2]
9   [2, 1, 2, 2, 2, 1, 1, 2, 1, 2]
10  [2, 1, 1, 1, 1, 2, 1, 2, 2, 1]
11  [2, 1, 6, 1, 1, 1, 2, 1, 1, 2]
12  [2, 1, 6, 2, 2, 2, 2, 1, 2, 1]
```

Figura 10: Archivo new_genotypes.txt con los cien miembros de la nueva generación

3.4.3. Evaluación de las nuevas generaciones

Para completar el experimento, se construye un mecanismo de automatización que asimila toda la caracterización previa de la red y guía el estudio de las nuevas generaciones engendradas a partir de la información contenida en new_genotypes.txt. Permite leer de este fichero y volcar la información de cada vuelta.

El experimento llevado a cabo desarrolla un total de veinte generaciones y se coordina a través del siguiente código en bash:

```
#!/bin/bash

# Generate 100 primitive trees with a reward associated
for ((i=1; i<=100; i++))
do
    python -m tfg.week_20240416.s00_tree_to_cyber # data.txt
done

# Cross the primitive trees to generate the first generation of trees
python -m tfg.week_20240416.crossover-mutate # new_genotypes.txt



# Evaluate the newly generated trees
for ((j=0; j<20; j++))
do
    echo "######## LAP $j #################" >> ./tfg/week_20240416/data2.txt # History with information of each lap

    for ((i=0; i<100; i++))
    do
        export VALOR=$i
        echo $i
        if python -m tfg.week_20240416.input_properties_dict; then
                python -m tfg.week_20240416.input_s00_tree_to_cyber # data2.txt & dataVuelta.txt
        else
            echo "ERROR - Impossible to evaluate";
        fi

    done

    python -m tfg.week_20240416.crossover-mutate-vuelta # new_genotypes.txt, From each lap, we generate new genotypes
    cat /dev/null > ./tfg/week_20240416/dataVuelta.txt # Clean the dataVuelta.txt file

done
```

Código 13: Configuración del primer experimento

Tras la finalización del experimento se obtiene un fichero con la primera generación aleatoria, un fichero con el último entrecruzado realizado y un fichero con el histórico de cada vuelta donde se podrá analizar la evolución en el valor de las recompensas y en la forma del genotipo de las redes.

A raíz de esta primera aproximación se han analizado dos mil ejecuciones del simulador, enfrentando tres veces al atacante contra cada uno de los fenotipos de red, por cada ejecución.

4. Resultados: Evaluación de la primera aproximación

Esta primera aproximación actúa como punto de partida del estudio. Se han aplicado los mecanismos definidos a lo largo del proyecto y se ha establecido una referencia inicial para observar posteriormente el progreso que suponen las siguientes medidas a aplicar.

Tal y como se menciona en la introducción, se trata de un análisis empírico y gracias al desarrollo e integración del modelo automatizado, se pueden evaluar los resultados y actuar en consecuencia.

A continuación, se muestra el análisis de esta primera aproximación y la aplicación de un primer mecanismo que mejora el resultado inicial.

El resultado de esta primera aproximación refleja una convergencia prematura a redes muy simples, que limitan al máximo la acción del atacante reduciendo el número de nodos que conforman las redes al mínimo posible. El experimento consta de veinte entrecruzados, pero desde la primera vuelta se comienzan a observar estructuras del tipo:

```
1 [2, 1, 1, x, x, x, x, x, x, x]
```

Código 14: Genotipo minimalista

que sustituyen muy rápido a cualquier otro genotipo dada su baja puntuación y cuyo fenotipo asociado es:

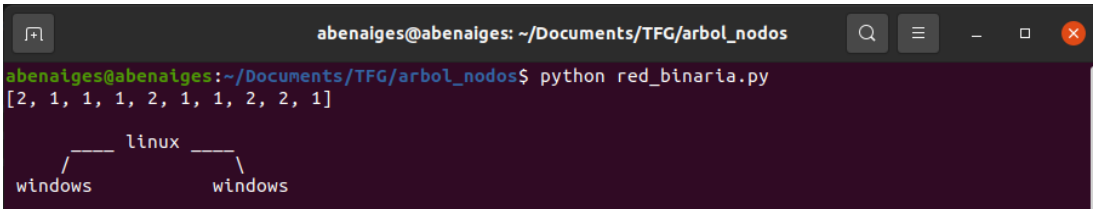

Figura 11: Estructura mínima de red

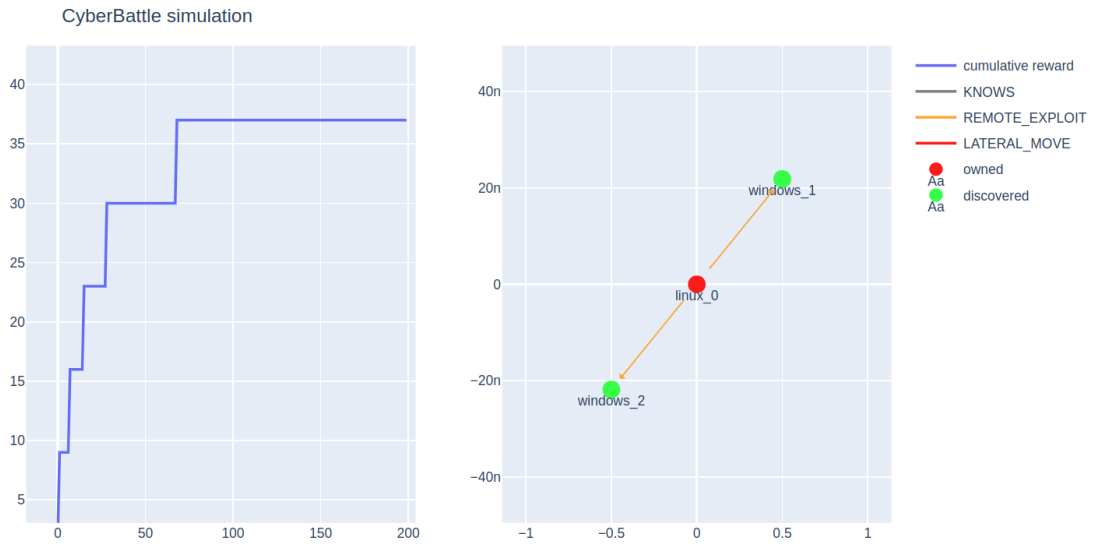

Figura 12: Estructura mínima de red. Representación a través de interfaz gráfica

Dado que la recompensa depende directamente de los movimientos que realiza el atacante, la solución óptima que encuentra el sistema, es restringir este movimiento. Este será el punto de partida de futuros análisis experimentales.

La evaluación derivada del planteamiento, indica que esta primera solución no es la más idónea ya que se aleja de un caso de uso real al ser reducida y poco realista.

Por esto último, tras un análisis del experimento, se propone una primera modificación para tratar de aumentar el número de nodos de la solución final.

Se decide relacionar directamente el número de nodos de la red con la recompensa que se obtiene. Es decir, a la hora de evaluar el rendimiento del agente atacante y asociar la recompensa al genotipo, que se divida por el número de nodos que ha podido recorrer el agente.

Adaptando el código original que define la clase que interpreta el genotipo, se consigue incorporar un contador de nodos terminales y no terminales de la red. Además, como se cuenta según se va construyendo, se puede determinar el total de nodos representables en forma de fenotipo a partir de cualquier genotipo.

```
"""
Count the possible number of nodes a binary tree can get,
using the given list of numbers.

Args:
    nums (list): A list of numbers representing the nodes of the binary
   tree.

Returns:
    Node: The root node of the binary tree.
    int: The number of terminal nodes.
    int: The number of non-terminal nodes.
"""
if not nums:
    return None, 0, 0

TerminalNodes = 0
NonTerminalNodes = 1

root = node(nums[0])

q = [root]
i = 1
while i < len(nums):
    if not q:  # Check if the queue is empty
        break
    curr = q.pop(0)
    if curr.val % 2 == 0:  # Nodo no terminal
        if i < len(nums):
            curr.left = node(nums[i])
            q.append(curr.left)

            if nums[i] % 2 == 0:
                NonTerminalNodes += 1
            else:
                TerminalNodes += 1
            i += 1
        if i < len(nums):
            curr.right = node(nums[i])
            q.append(curr.right)

            if nums[i] % 2 == 0:
                NonTerminalNodes += 1
            else:
                TerminalNodes += 1
            i += 1

return root, TerminalNodes, NonTerminalNodes
```

Código 15: Función para contabilizar los nodos que conforman una red

Esta modificación aportará valor a la relación que se establece entre el número de nodos y la recompensa total que se puede obtener. Existe diferencia entre obtener una recompensa baja cuando el número de nodos es bajo y obtener una recompensa baja

cuando el número de nodos es mayor y, por tanto, el número de acciones que puede tomar el agente atacante también lo es.

Una vez se ejecuta de nuevo el experimento, se observa que esta modificación retrasa, en número de episodios, la posibilidad alcanzar esta convergencia. En este caso, la referencia inicial será la vuelta cuatro y la mejoría se observa hasta la vuelta diez.

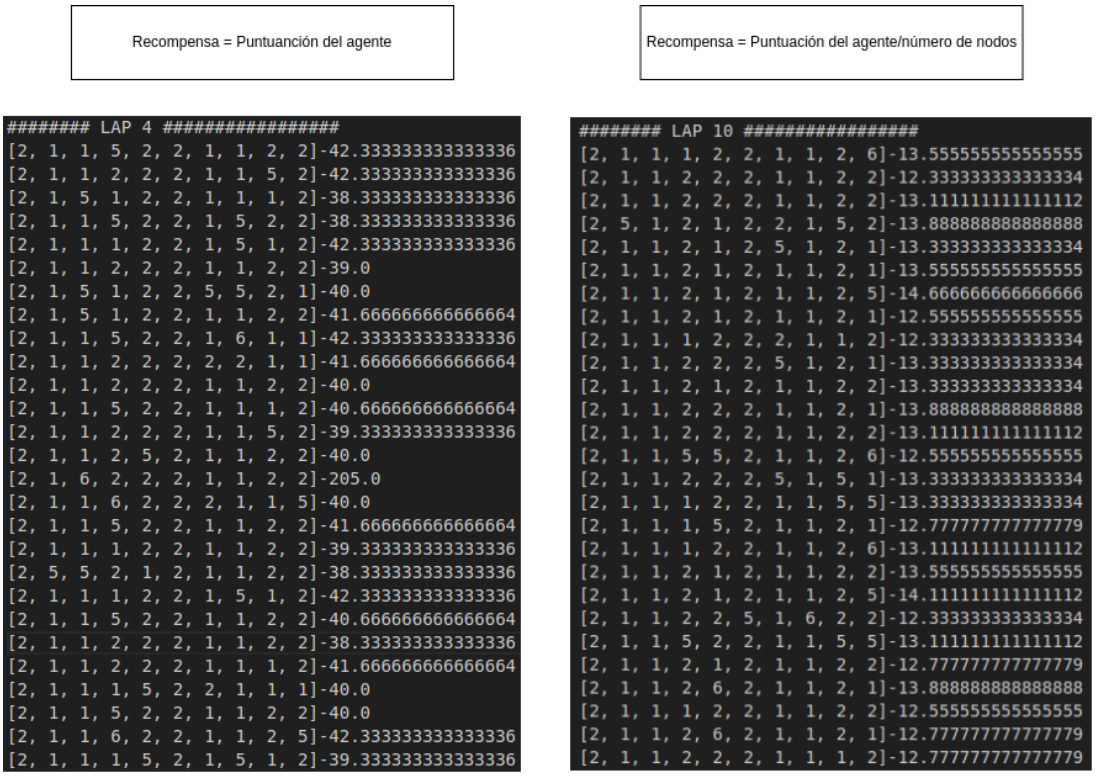

Figura 13: Comparativa de resultados dividiendo por el número de nodos posible de cada red

Así, con una leve modificación a la hora de evaluar la recompensa obtenida, se genera un impacto notable en la solución final.

A raíz de este primer cambio propuesto se ha tratado de explotar la relación *recompensa obtenida - recompensa potencial* observando como influye en el numero de redes de genotipo minimalista que se obtiene, para valorar el impacto de esta modificación. De esta forma se vuelve a evaluar el caso inicial obteniendo el punto de partida en la vuelta ocho, lo que demuestra también una alta volatilidad de los resultados. Sería necesario entonces, imponer unas mejores restricciones.

A continuación se muestra una gráfica en forma de histograma que refleja a partir de qué vuelta se obtienen más de 90 % de redes con genotipo minimalista en función del factor que multiplica el divisor de la operación división de recompensa entre el número de nodos que puede recorrer el agente.

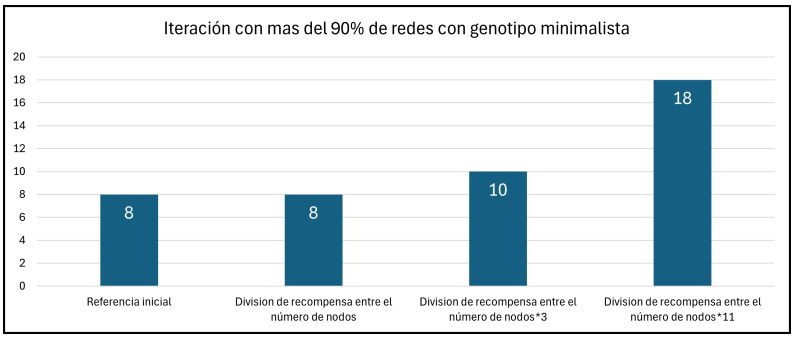

Figura 14: Experimentos explotando relación *recompensa obtenida - recompensa potencial*

Se puede observar el impacto que genera multiplicar el divisor de la relación. No obstante el porcentaje de redes con genotipo minimalista supera el 70 % desde la 2ª o 3ª vuelta en todos los casos. Esto hace pensar que quizás se debe a que el número de nodos totales también es muy reducido.

Por capacidad de cómputo dotar de más nodos a los fenotipos, supone que es necesario limitar el número total de ejecuciones por experimento. Hasta ahora, cada experimento consta de más de dos mil ejecuciones totales.

Haciendo otra comprobación se limita la producción inicial de genotipos a diez en vez de cien genotipos con treinta nodos cada uno y los resultados son los siguientes.

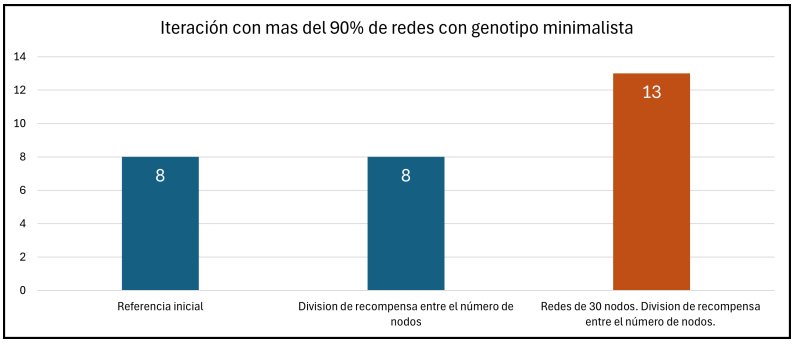

Figura 15: Comparativa ampliando el número de nodos a treinta

Se puede visualizar que calculando la misma relación recompensa entre numero de nodos, habiendo limitado el número total posible a treinta en vez de a diez afecta positivamente en el número de vueltas que se necesitan para alcanzar un 90 % de redes con genotipo minimalista.

Por otro lado, viendo las diferentes topologías que se van analizando se pueden observar genotipos que reflejan que ciertas estructuras son más complejas de recorrer que otras.

Alejándose del genotipo que produce una convergencia prematura, surgen diferencias entre estructuras conformadas por el mismo número de nodos, como por ejemplo:

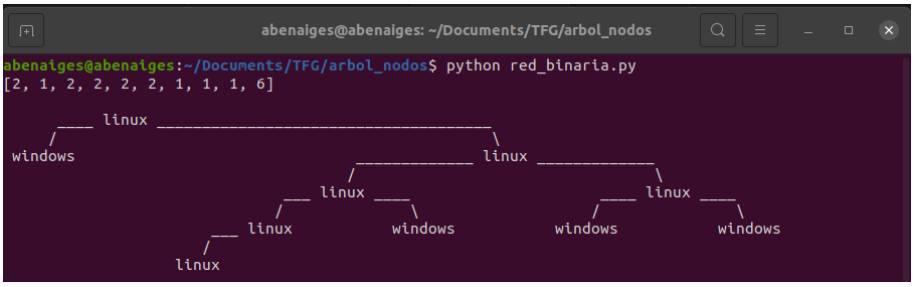

Figura 16: Red de 10 nodos. Recompensa asociada: 8.61667

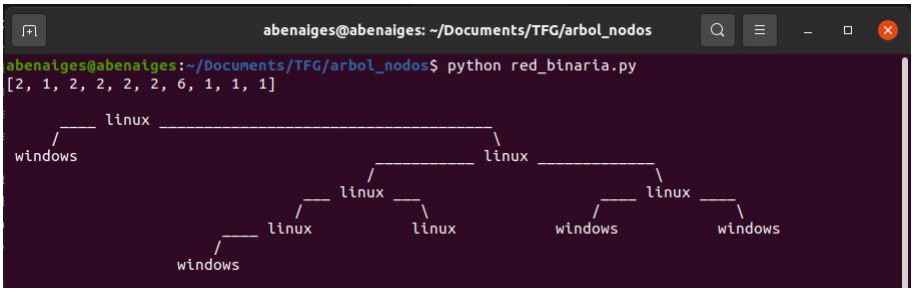

Figura 17: Red de 10 nodos. Recompensa asociada: 24.11667

Ambas estructuras son altamente similares, pero esa diferencia de puntuación induce a pensar que el orden de los nodos no terminales que quedan vacíos, influye.

A partir de este punto, queda continuar analizando y evaluando la aplicación de nuevos mecanismos que acerquen el estudio a la hipótesis inicial de generar una red de topología, difícil de descubrir, recorrer e infectar para el agente atacante basado en DQN.

5. Conclusiones

CyberBattleSim, siendo una herramienta muy potente para simular y analizar abstracciones de escenarios reales, es limitado a la hora de realizar experimentación con conjuntos de datos, ya que requiere de una configuración exhaustiva y compleja para replicar una situación real. Además, no dispone de mecanismos que permitan la evaluación y el entrenamiento de modelos basados en conjuntos de datos que requieren de la automatización de experimentos.

La propuesta, pasa por integrar mecanismos para realizar experimentos automatizados y sienta como base un primer experimento que supone una modificación significativa en la forma en que se estructura la red de nodos en CyberBattleSim.

Habiendo conseguido evaluar grandes conjuntos de datos haciendo uso del simulador, se descubren limitaciones al plantear un modelo de generación de redes antagónico, basado en evolución gramatical.

- El desarrollo evolutivo genera genotipos minimalistas.
- La caracterización de los nodos es poco realista en términos de similitud con redes empresariales reales.
- El rendimiento computacional que requiere la experimentación es alto.

Estas limitaciones sientan las bases para nuevas líneas de investigación dentro del marco que desarrolla dicho modelo.

Así, se plantean las siguientes acciones futuras:

- Multiplicar por un índice el número de nodos a la hora de dividir, para que aumente su impacto a la hora de realizar la división con la recompensa. Visto que esta modificación influye sobre los resultados, es interesante seguir explotándola.
- Realizar una selección elitista de los genotipos con peor puntuación asociada y mantenerlos para la siguiente generación sin entrecruzar. Esto ayudaría a garantizar que las soluciones de alta calidad no se pierdan en generaciones posteriores y pudieran seguir contribuyendo a la evolución de la especie.
- Incorporar nuevas vulnerabilidades a los nodos Linux y Windows más realistas o actualizadas. En este momento, el *crackeo* de un gestor de contraseñas y el escaneo de comandos de *Bash* pueden ser demasiado sencillas si se considera modelar un sistema actual. No obstante es importante reconocer cómo se configuran y la información que aportan, para garantizar la estructura de la red y tenerlo en cuenta a la hora de incorporar nuevas vulnerabilidades.
- Modificar de la condición terminal/no terminal de los nodos; hasta ahora, el sistema operativo se relaciona directamente con la condición terminal/no terminal del nodo. En un futuro, sería interesante independizar la característica estructural, del tipo de nodo basado en su sistema operativo. Es decir, decidir que los nodos fueran terminales o no, independientemente del servicio que albergan.

- Por último, los requisitos computacionales del experimento fuerzan a realizar redes reducidas imponiendo un límite pequeño de nodos. En un futuro, sería interesante aumentar el número de miembros del genotipo para representar redes de más de diez conexiones.

Revisando los objetivos del proyecto, se puede concluir que:

- Se ha conseguido entender y hacer uso del marco de trabajo de CyberBattleSim
- Se ha integrado el método que permite la experimentación.
- Se ha incorporado un primer experimento que sienta una referencia inicial.
- Se ha automatizado el proceso de iteración para la evaluación de resultados
- Se ha realizado una evaluación de la primera aproximación de las que se han podido obtener líneas de acción futuras.

Una vez se ha completado el experimento, queda demostrada la utilidad del modelo propuesto. Aunque sencillo, el caso de uso refleja las posibilidades que permite el entorno de simulación CyberBattleSim y el sinfín de variaciones que dicho marco de trabajo admite para tratar de configurar escenarios muy parejos a la realidad.

En el caso de este estudio, cabe destacar que la utilización de la gramática con la que se estipulan unas reglas de producción, ofrece un modelado de datos versátil y transformable. Gracias a las reglas definidas, se impone una serie de condiciones que consiguen llevar a cabo operaciones con estructuras complejas de datos. Las condiciones garantizan que estas estructuras no se corromperán con las posibles modificaciones que puedan surgir durante el procedimiento.

Además, el modelo de programación evolutiva antagónica, desde un inicio presenta grandes posibilidades y un abanico de posibles cambios e incorporaciones a añadir para mejorar los resultados obtenidos a través del experimento desarrollado en este trabajo.

Referencias

[1] Microsoft Defender Research Team. *CyberBattleSim*. https://github.com/microsoft/cyberbattlesim. Created by Christian Seifert, Michael Betser, William Blum, James Bono, Kate Farris, Emily Goren, Justin Grana, Kristian Holsheimer, Brandon Marken, Joshua Neil, Nicole Nichols, Jugal Parikh, Haoran Wei. 2021.

[2] Microsoft Security Team. *Gamifying Machine Learning for Stronger Security and AI Models*. 2021. URL: https://www.microsoft.com/en-us/security/blog/2021/04/08/gamifying-machine-learning-for-stronger-security-and-ai-models/.

[3] UM. O'Reilly et al. «Adversarial genetic programming for cyber security: a rising application domain where GP matters.» En: (2020). DOI: 10.1007/s10710-020-09389-y.

[4] Marina de la Cruz Echeandia. «Evolución gramatical y semántica». En: (2022). URL: https://repositorio.uam.es/bitstream/handle/10486/4886/31769_cruz_echeandia_marina_de_la.pdf?sequence=1&isAllowed=y.

[5] Riccardo Poli, William B. Langdon y Nicholas Freitag McPhee. *A Field Guide to Genetic Programming*. Publisher, 2008. ISBN: 978-1-4092-0073-4.

[6] Michael O'Neill y Conor Ryan. *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. Kluwer Academic Publishers, 2003. ISBN: 978-1-4020-7444-8.

[7] @enlightenment609. *Gramatical Evolution*. 2022. URL: https://www.youtube.com/watch?v=vAA0Q0bw1yg.

[8] Pablo Esaú Mejía Medina. «Programación Genética y Evolución Gramatical : un enfoque combinado usando Straight Line Programs». En: *Repositorio de la Universidad de Cantabria* (jun. de 2019). Ed. por José Luis Montaña Arnaiz. Atribución-NoComercial-SinDerivadas 3.0 España. URL: http://hdl.handle.net/10902/16926.

[9] Christian Fisher et al. «Multi-agent reinforcement learning in cyberbattlesim». En: (2022). URL: https://cradpdf.drdc-rddc.gc.ca/PDFS/unc425/p816591_A1b.pdf.

[10] Greg Brockman et al. *OpenAI Gym*. 2016. eprint: arXiv:1606.01540.

[11] Prafulla Dhariwal et al. *OpenAI Baselines*. https://github.com/openai/baselines. 2017.

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

[13] Markel Sanz. *Introducción al aprendizaje por refuerzo - Parte 2: Q-Learning*. 2022. URL: https://markelsanz14.medium.com/introducci%C3%B3n-al-aprendizaje-por-refuerzo-parte-2-q-learning-883cd42fb48e.

[14] J. Esteban, Department of Electrical Engineering y Computer Science MIT. «Simulating Network Lateral Movements through the CyberBattleSim Web Platform». En: *MIT DSpace* (2022). URL: https://dspace.mit.edu/bitstream/handle/1721.1/143191/Esteban-jesteban-meng-eecs-2022-thesis.pdf?sequence=1&isAllowed=y.

[15] Clinton Sheppard. *Genetic Algorithms with Python*. CreateSpace Independent Publishing Platform (April 29, 2016), 2016. ISBN: 1540324001.

6. Anexo

Se incluyen, a continuación en este anexo, partes del código desarrollado que aportan valor pero facilita el trabajo del lector.

```
# Linux node
node.node_info = m.NodeInfo(
    agent_installed=(node == self.tree.root),
    services=[
        m.ListeningService("HTTPS"),
        m.ListeningService(
            "SSH", allowedCredentials=[ssh_password(node.unique_label)]
        ),
    ],
    firewall=m.FirewallConfiguration(
        incoming=chain_sample.DEFAULT_ALLOW_RULES,
        outgoing=chain_sample.DEFAULT_ALLOW_RULES,
    ),
    value=100,
    properties=["MySql", "Ubuntu", "nginx/1.10.3"],
    owned_string="Intermediate chain node owned, no intinsic value",

    vulnerabilities=dict(
        ProbeLinux=m.VulnerabilityInfo(
            description="Probe to check if the node runs Linux",
            type=m.VulnerabilityType.REMOTE,
            outcome=m.ProbeSucceeded(["Ubuntu"]),
            reward_string="Remote machine is running Linux",
            cost=5.0,
        ),
        ProbeWindows=m.VulnerabilityInfo(
            description="Probe to check if the node runs Windows",
            type=m.VulnerabilityType.REMOTE,
            outcome=m.ProbeFailed(),
            reward_string="Remote machine is not running Windows",
            cost=0,
        ),
        ScanBashHistory=m.VulnerabilityInfo(
            description="Scan bash history for possible references to
   other machines",
            type=m.VulnerabilityType.LOCAL,
            outcome=m.LeakedNodesId(leaked_nodes_ids),
            reward_string="Found a reference to a remote Windows node in
    bash history",
            cost=10.0,
        ),
        ScanExplorerRecentFiles=m.VulnerabilityInfo(
            description="Scan Windows Explorer recent files for possible
    references to other machines",
            type=m.VulnerabilityType.LOCAL,
            outcome=m.ExploitFailed(),
            reward_string="Trap: feature not supported on Linux",
            cost=0,
        ),
        SudoAttempt=m.VulnerabilityInfo(
```

```
            description="Attempt to sudo into admin user",
            type=m.VulnerabilityType.LOCAL,
            outcome=m.ExploitFailed(),
            reward_string="Trap: suspicious attempt to run sudo",
            cost=0,
        ),
        **vulnerabilities_left,
        **vulnerabilities_right,  # spreads other dictionary
    ),
)
```

Código 16: Servicios y vulnerabilidades del nodo Linux

```
# windows node
node.node_info = m.NodeInfo(
    services=[],
    value=100,
    properties=["Windows", "Win10", "Win10Patched"],
    vulnerabilities=dict(
        ProbeLinux=m.VulnerabilityInfo(
            description="Probe to check if the node runs Linux",
            type=m.VulnerabilityType.REMOTE,
            outcome=m.ProbeFailed(),
            reward_string="Remote machine is not running Linux",
            cost=0,
        ),
        ProbeWindows=m.VulnerabilityInfo(
            description="Probe to check if the node runs Windows",
            type=m.VulnerabilityType.REMOTE,
            outcome=m.ProbeSucceeded(["Windows"]),
            reward_string="Remote machine is running Windows",
            cost=25.0,
        ),
        ScanBashHistory=m.VulnerabilityInfo(
            description="Scan bash history for possible references to
   other machines",
            type=m.VulnerabilityType.LOCAL,
            outcome=m.ExploitFailed(),
            reward_string="Trap: feature not supported on Windows!",
            cost=0,
        ),
        ScanExplorerRecentFiles=m.VulnerabilityInfo(
            description="Scan Windows Explorer recent files for possible
    references to other machines",
            type=m.VulnerabilityType.LOCAL,
            outcome=m.ExploitFailed(),
            reward_string="Not Found a reference to a remote Linux node
   in bash history",
            cost=0,
        ),
        SudoAttempt=m.VulnerabilityInfo(
            description="Attempt to sudo into admin user",
            type=m.VulnerabilityType.LOCAL,
            outcome=m.ExploitFailed(),
            reward_string="Trap: feature not supported on Windows!",
```

```
            cost=0,
        ),
        # **vulnerabilities,
    ),
)
```

Código 17: Servicios y vulnerabilidades del nodo Windows

```
leaked_nodes_ids = []
vulnerabilities_left = {}
vulnerabilities_right = {}

if node.left is not None:
leaked_nodes_ids.append(node.left.unique_label)

if node.left.val % 2 == 0:
    vulnerabilities_left["CrackKeepPass_left"] = m.VulnerabilityInfo(
        description="Attempt to crack KeepPass and look for credentials",
        type=m.VulnerabilityType.LOCAL,
        outcome=m.LeakedCredentials(
            credentials=[
                m.CachedCredential(
                    node=node.left.unique_label,
                    port="SSH",
                    credential=ssh_password(node.left.unique_label),
                )
            ]
        ),
        reward_string=f"Discovered password to Linux machine {node.left.unique_label}",
        cost=50.0,
    )
else:
    vulnerabilities_left["CrackKeepPassX_left"] = m.VulnerabilityInfo(
        description="Attempt to crack KeepPassX and look for credentials",
        type=m.VulnerabilityType.LOCAL,
        outcome=m.LeakedCredentials(
            credentials=[
                m.CachedCredential(
                    node=node.left.unique_label,
                    port="RDP",
                    credential=rdp_password(node.left.unique_label),
                )
            ]
        ),
        reward_string=f"Discovered password to Windows machine {node.left.unique_label}",
        cost=50.0,
    )
```

Código 18: Vulnerabilidad en el gestor de contraseñas (CrackKeepPass Vulnerabilities)

```
if node.right is not None:
leaked_nodes_ids.append(node.right.unique_label)

if node.right.val % 2 == 0:
    vulnerabilities_right["CrackKeepPass_right"] = m.VulnerabilityInfo(
        description="Attempt to crack KeepPass and look for credentials",
        type=m.VulnerabilityType.LOCAL,
        outcome=m.LeakedCredentials(
            credentials=[
                m.CachedCredential(
                    node=node.right.unique_label,
                    port="SSH",
                    credential=ssh_password(node.right.unique_label),
                )
            ]
        ),
        reward_string=f"Discovered password to Linux machine {node.right.unique_label}",
        cost=50.0,
    )
else:
    vulnerabilities_right["CrackKeepPassX_right"] = m.VulnerabilityInfo(
        description="Attempt to crack KeepPassX and look for credentials",
        type=m.VulnerabilityType.LOCAL,
        outcome=m.LeakedCredentials(
            credentials=[
                m.CachedCredential(
                    node=node.right.unique_label,
                    port="RDP",
                    credential=rdp_password(node.right.unique_label),
                )
            ]
        ),
        reward_string=f"Discovered password to Windows machine {node.right.unique_label}",
        cost=50.0,
    )
```

Código 19: Vulnerabilidad en el gestor de contraseñas (CrackKeepPass Vulnerabilities)(II)