ISSN 2215-5910
Investigación

APRENDIZAJE POR DEFENSA REACTIVA: EL NUEVO MODELO DE ENTRENAMIENTO CONTRA MALWARE

Bernal Rojas Villalobos
César Rodríguez Bravo

RESUMEN

Este artículo propone un nuevo método de defensa en ciberseguridad utilizando inteligencia artificial, que puede ser aplicado en diferentes entornos, desde equipos de escritorio (personales y empresariales), hasta equipos móviles (tabletas, laptops e incluso celulares). Los sistemas antimalware o antivirus tradicionales funcionan de manera determinística (es decir, basados en reglas), mientras que los cibercriminales aparecen con técnicas cada vez más creativas para burlar estos sistemas. En este artículo se propone el diseño de una **arquitectura de inteligencia artificial** capaz de detectar un código malicioso mediante aprendizaje automático, con la capacidad de identificar intentos creativos de burlar los sistemas de seguridad.

Esta arquitectura se basa en el entrenamiento de dos inteligencias artificiales, una maliciosa o atacante (hacker) y otra defensiva. El entrenamiento de ambos modelos se realiza mediante aprendizaje por refuerzo con **Q-Learning y redes neuronales convolucionales**, respectivamente. La red neuronal atacante aprenderá mediante aprendizaje por refuerzo a ejecutar los métodos más utilizados por los hackers para burlar los sistemas antivirus o antimalware, mientras que el modelo defensor hará uso de **Deep Learning** para **convertir los archivos binarios maliciosos en imágenes,** y así poder entrenarse para identificar los patrones del malware y cualquier otra variante generada por el atacante.

Palabras claves: ciberseguridad, inteligencia artificial, agentes inteligentes, antivirus, captura la bandera.

ABSTRACT

This article proposes a new method of defense in cybersecurity using artificial intelligence that can be applied in various environments, from desktop computers (personal and business), to mobile computers (tablets, laptops and even cell phones). Traditional anti-malware or anti-virus systems work deterministically (based on rules), while cybercriminals come up with increasingly creative techniques to bypass these systems. This article proposes the design of an **artificial intelligence** architecture capable of detecting malicious code through machine learning, with the ability to identify creative attempts to bypass security systems.

This architecture is based on the training of two artificial intelligences, one malicious or attacking (hacker) and the other defensive. The training of both models is carried out by means of reinforcement learning with **Q-Learning and convolutional neural networks**,

Bernal Rojas es especialista en Ciberseguridad y estudiante de Ingeniería en Ciencia de Datos en Lead University y César Rodríguez es Profesor de Lead University, Máster en Ciberseguridad e Inventor con más de 100 aplicaciones a patentes en Estados Unidos, Europa y China

respectively. The attacking neural network will learn through reinforcement learning to execute the methods most used by hackers to circumvent antivirus or antimalware systems, while the defender model will use **Deep Learning** to **convert malicious binary files into images** and thus be able to train to identify malware patterns and any other variant generated by the attacker.

Key words: cybersecurity, artificial intelligence, intelligent agents, antivirus, capture the flag.

INTRODUCCIÓN

Cuando se escucha hablar sobre ciberseguridad se piensa que es algo que concierne solamente a una empresa. Y es común pensar que, como personas, somos insignificantes para ser el objetivo de un atacante. Pero eso es totalmente falso, los objetivos de un cibercriminal pueden ser individuos como cualquiera de nosotros.

Al final la mayoría tenemos cuentas bancarias, información sensible en nuestras computadoras, contraseñas, o simplemente queremos que nuestra privacidad no se vea comprometida.

Es por esta razón que normalmente se tiende a utilizar algún sistema antimalware o antivirus. Cuando se habla de malware se refiere a un código malicioso o malintencionado. En las grandes corporaciones existen múltiples controles técnicos que endurecen la tarea para un cibercriminal, pues al implementar tantos métodos de defensa, no es sencillo para un atacante lograr su objetivo.

Sin embargo, con el COVID-19 que está aumentando y acelerando la cantidad de empleados trabajando desde sus casas. Es de esperar, una explosión de ataques enfocados en los equipos domésticos.

El problema es que muchos no tienen en su casa algún mecanismo o sistema de ciberseguridad pues, como todo, tiene un costo y no es necesariamente bajo.

Es por esta razón que se recomienda al menos utilizar como mínimo, algún sistema antivirus, existente en el mercado.

Ahora bien, la mayoría de estos antivirus funcionan mediante la firma de código malicioso (o técnica similar). El problema que esto sugiere (como veremos más adelante) es que los atacantes han desarrollado formas en las que pueden burlar fácilmente estos controles al hacer pequeños cambios en el código u otras técnicas un poco más sofisticadas y lograr ejecutar el código malicioso, causando el daño en el equipo, comprometiendo la privacidad o su objetivo en particular.

Surge la necesidad de un sistema capaz de reconocer todos estos pequeños cambios que podría realizar un atacante para burlar un sistema de seguridad. Pero esto es complicado y es un reto, pues al final de cuentas el hacking se ha convertido en un juego de creatividad. Entre más creativa sea una técnica, mayor probabilidad de pasar por desapercibida ante un antivirus.

Por esta razón, en este artículo se sugiere que,

"La creatividad, se combate con creatividad".

De esta forma, ahora nos encontramos ante un reto de diseño ¿Cómo diseñar un sistema creativo?

La respuesta sugiere inteligencia artificial pero, ahora bien, la mayoría de los algoritmos de *machine learning* son deterministas también y se entrenan a partir de información previamente conocida, por ello, cualquier variante que haya quedado por fuera del conjunto de entrenamiento, será completamente desconocida para el sistema.

Por esta razón se debe buscar alguna solución que permita generar múltiples escenarios de variantes que de alguna forma puedan tener una cierta parte de creatividad.

En resumen, sería necesario que el modelo defensor aprenda de un hacker. El problema técnico que esto presenta es la cantidad de horas que necesita de supervisión humana. Donde el hacker genere una gran cantidad de modificaciones para que el modelo defensor pueda aprender a reconocer esas variaciones.

Como solución, se propone aprendizaje por refuerzo ya que el RL (reinforcement learning) resulta ser un método que permite que un agente inteligente aprenda a encontrar soluciones complejas, de acuerdo con su conjunto de acciones, logrando así, la creatividad buscada.

En pocas palabras, se entrenará un agente inteligente, que aprenda a "hackear" y genere de forma automática, muchas de esas variantes en el código malicioso.

Para este artículo se parte del hecho de que el *malware* ya ha sido introducido en el equipo. No importa el vector de ataque, es decir, la forma por la que haya ingresado el malware o código malicioso, ya que este factor no será relevante para esta investigación.

Lo realmente importante son las técnicas que se utilizan para burlar los sistemas antivirus. Por ello se establece la base de los experimentos en una computadora con un sistema antivirus activo donde se ha logrado introducir un malware que aún no ha sido ejecutado.

En este caso, la tarea del agente malicioso es aprender a generar variaciones para lograr ejecutar el *malware* con éxito, burlando el antivirus; mientras que el agente defensor debe aprender de las variantes generadas por el agente malicioso para identificar rápidamente el intento de ejecución de un código malicioso alterado intentando burlar el sistema antivirus.

CONTEXTO

Esta sección provee un conocimiento importante para el entendimiento de los conceptos de este artículo y términos que se estarán utilizando. Si esta sección no es de su interés, puede saltar directamente a la sección *Método propuesto*.

Antimalware

Los sistemas antimalware o más popularmente conocidos como antivirus, aunque no es del todo correcto llamarlos así, pues el virus informático es solamente un tipo de código malicioso de la gran cantidad de tipos existentes. La principal característica de un virus informático es que necesita estar pegado a otro programa que se esté ejecutando. Como cuando se abre una imagen que contiene un virus, durante el tiempo que esa imagen esté abierta, el virus podrá trabajar. Pero cuando se cierre, dejará de funcionar.

Por otra parte, existen otros tipos de malware como los gusanos informáticos, que no necesitan de otro programa. Ellos tienen la capacidad de moverse por la red, de manera autónoma.

Cualquier usuario de un sistema antimalware, esperaría que su sistema de seguridad sea capaz de detectar cualquier programa malicioso, ya sea un virus, un gusano o cualquier otro tipo de malware existente. Por esta razón la forma correcta es llamarlos es **sistema antimalware**.

La finalidad de estos sistemas es detectar y bloquear cualquier código que tenga una intención de alterar el funcionamiento de nuestro equipo para algún objetivo malintencionado. Sin embargo, esta no es una tarea sencilla, ya que al final para una computadora cualquier acción que se ejecute se hace de manera independiente y no podría saber si su intención es buena o mala.

Para ello es que se han adoptado varias técnicas que permitan detectar software que quiera causar algún daño, como veremos a continuación.

Firma de código: Esta es la forma más habitual y común que se encuentra en los sistemas comerciales gratis, de pago con versiones domésticas o los paquetes que no corresponden a servicios empresariales. Funcionan con base en la identificación de código malicioso ya previamente conocido.

Si alguna vez usted ha escuchado *"la base de datos del antivirus, ha sido actualizada"*, esto se debe a que día a día los proveedores de estos servicios actualizan sus bases de datos con nuevas firmas de código malicioso. En el momento que se quiera ejecutar un programa, su computadora va a calcular un hash, esto es un algoritmo matemático al que se le ingresa el código del programa que se quiere ejecutar y devuelve un valor único para ese código, de este modo no existen dos salidas iguales para un mismo código.

La comparación de hashes es como identificar a un virus por su huella digital.

El sistema va a buscar el hash del programa que se desea ejecutar en su base de datos de malware y si encuentra uno igual, lo va a detectar cómo malicioso. Como se muestra en la figura 1.

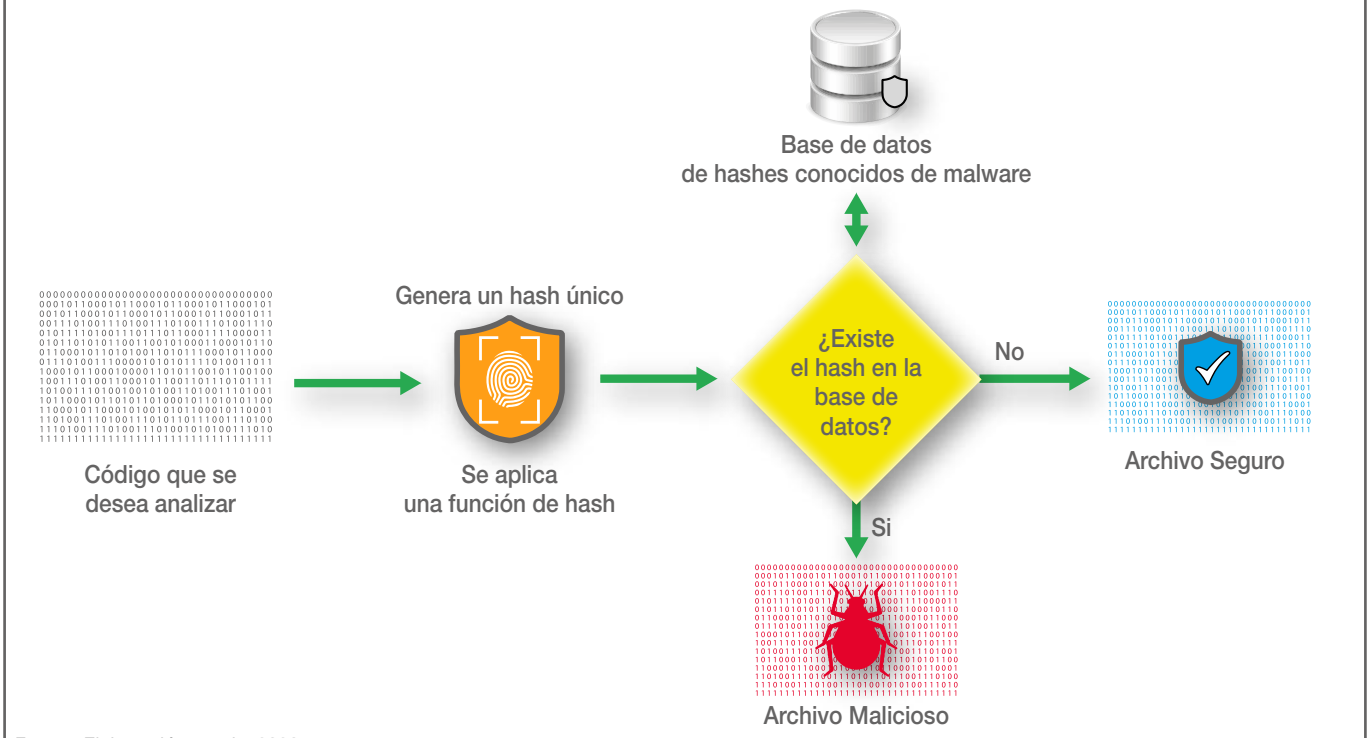
Detección por comportamiento: Esta es una técnica que hace un escaneo del sistema en el momento que se detecta un funcionamiento erróneo o atípico. Es más útil para detectar malware no conocido, pero es más riesgoso, pues se ejecuta una vez el sistema ya se ha infectado.

Sandbox o caja de arena: Esta es una técnica que crea un entorno aislado del sistema real, cualquier cosa que pueda suceder dentro de este entorno, no afectará el sistema físico o sistema real del equipo.

De este modo, dicha técnica ejecuta el software y analiza los procesos y comportamiento que tengan para identificar posibles comportamientos maliciosos.

Esta técnica es muy efectiva para detectar malware, sin embargo, son muy lentas, pues deben hacer todo este proceso para cualquier aplicación que se desee ejecutar.

FIGURA 1. DIAGRAMA DE FUNCIONAMIENTO DE UN ANTIVIRUS DE DETECCIÓN POR FIRMA DE CÓDIGO



Fuente: Elaboración propia, 2020.

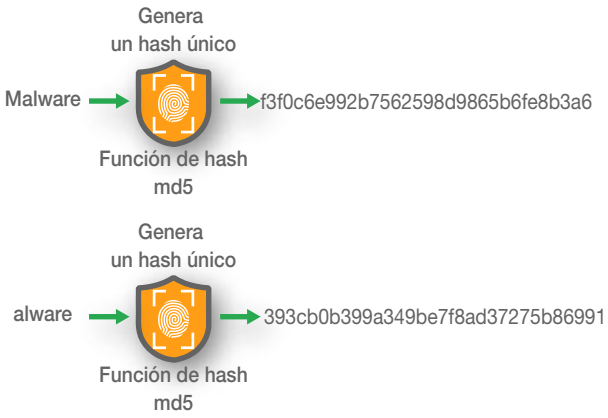
Por lo general esta técnica no está implementada en los paquetes para uso doméstico, siendo que constituyen extras que se compran o reservan para paquetes empresariales.

Técnicas de evasión de sistemas antimalware

Al conocer el funcionamiento de los principales sistemas antimalware, los atacantes pueden idear formas creativas de crear un "bypass" o burlar el sistema antimalware. A continuación, se exponen las técnicas más usadas para cada método de funcionamiento anteriormente expuesto.

Firma de código: a pesar de ser el método más común, es el más vulnerable a ser evadido, pues su seguridad radica en que el código malicioso que se quiere ejecutar, el cual debe ser exactamente igual al código ya conocido en la base de datos. Si se recuerda cuando se mencionó cómo funciona un hash, se dijo que cada código, tiene una salida única, algo como una huella digital. Es decir, con un pequeño cambio en el código, la salida será completamente diferente (ver figura 2). Entonces, si se cambian palabras en el código fuente malicioso, la salida será completamente diferente a la del código malicioso ya conocido, y entonces cuando se realiza la comparación de los hashes, estos no serán iguales.

FIGURA 2. COMPARACIÓN DE SALIDAS DE FUNCIÓN DE HASH, DE ACUERDO CON UN CAMBIO PEQUEÑO EN SU ENTRADA



Fuente: Elaboración propia, 2020.

En el ejemplo anterior, la palabra "malware" al aplicarle un algoritmo de hash llamado Sha256, devuelve un valor único.

Si se elimina la letra "m", dejando únicamente la palabra "alware", se obtiene otro valor completamente distinto.

Cuando el antivirus quiera comparar ambas firmas o valores hash, serán diferentes y lo clasificará como un código seguro. Esta es la forma más habitual de burlar estos sistemas.

Ahora bien, hay técnicas más avanzadas, como malware polimórfico que tiene la capacidad de cambiar su propio código fuente. También se utiliza mucho la inserción de caracteres aleatorios, entre otras formas de hacer que el hash cambie para que, cuando se compare, no sea el igual a las muestras ya conocidas.

Detección por comportamiento: Este método tiene una gran desventaja que lo hace poco recomendable. Es el hecho de que deban ejecutar los programas (virus) primero. Si uno de estos programas era un código mal intencionado podría dañar el equipo antes de ser detectado. Por esta razón, en sí mismo es una vulnerabilidad y no es recomendable de utilizar.

Sandbox o caja de arena: Esta es la forma más efectiva de controlar el ingreso de un malware al sistema, sin embargo, esta tecnología es realmente lenta, por lo cual sugiere un problema en su uso, pues hará muy lenta la labor diaria. Si bien es cierto, es la más efectiva, el hecho de que sea tan lenta y computacionalmente costosa, adicionada al hecho de que es poco común encontrarla en computadores domésticos, hace que no sea un método por el cual se frenen muchos ataques.

Inteligencia Artificial

Primero vamos a definir a la inteligencia artificial como la capacidad que tiene una máquina de emular el comportamiento cognitivo de un ser humano a partir de una señal del entorno. Ahora bien, por definición se sabe que la máquina para poder hacer esto, debe aprender cómo resolver esa tarea en particular. Para lograr esto se utilizan varias ramas de aprendizaje, dentro de ellas:

Machine Learning: el aprendizaje automático por lo general trabaja con datos estructurados. Son algoritmos matemáticos que tienen la capacidad de encontrar patrones relevantes para ajustar sus hiper-parámetros o valores internos para modelar una predicción. De modo que, para cualquier nueva entrada, pueda predecir una salida en base a los datos que utilizó para su entrenamiento.

Deep Learning: El aprendizaje profundo es una serie de algoritmos de aprendizaje de máquina que, por lo general, están enfocados a resolver problemas de datos no estructurados, o datos multimedia. Es decir, son particularmente buenos resolviendo tareas con imágenes, audios, videos, texto, entre otros tipos de multimedia.

Reinforcement Learning: El aprendizaje por refuerzo es una serie de algoritmos que se pueden llamar algoritmos evolutivos, pues las ramas anteriores necesitan un conjunto de datos para su entrenamiento, mientras que este aprende a medida que pasa el tiempo, obteniendo una realimentación del entorno en cada iteración.

Se ha logrado demostrar que estos modelos tienen un alto impacto y eficacia, como el caso de Alpha Go [3], una inteligencia artificial que aprendió a jugar Go y le ganó al campeón del mundo.

Redes Neuronales Convolucionales

Es un tipo de Red Neuronal Artificial, que trabaja sobre matrices multidimensionales, a través de filtros y capas ocultas. Las neuronas artificiales de estos modelos corresponden a campos receptivos, tal y como funciona en la corteza visual, lo cual hace que sean especialmente buenos para resolver tareas de visión artificial.

Q-Learning

Es un algoritmo RL (reinforcement learning) de diferencia temporal, esto significa que el algoritmo estima la función de valor de acción $q(st, at)$ a partir de una suposición inicial, y actualiza paso a paso su estimación con referencia al valor de estados y acciones futuros. Q-learning es capaz de aprender a ejecutar una acción óptima mientras explora el entorno.

Formalmente, dado un problema de RL $\langle S,A,T,R \rangle$, donde

- *S:* es el conjunto de todos los estados del entorno.
- *A:* es el conjunto de todas las acciones disponibles para el agente.
- *T:* $P(st + 1 \mid st,at)$ es la función de transición del entorno o la probabilidad de que el entorno pase de estado *st* al estado $st + 1$ dado una acción tomada por el agente *at*. Es decir, la probabilidad de que se cambie de un estado al otro.
- *R:* $P(rt \mid st,at)$ es la función de recompensa o la probabilidad de que el agente reciba una recompensa *rt* dado una acción tomada por el agente *at* en el estado *st*. Es decir, la probabilidad de que el agente reciba una recompensa, si ejecutó una tarea correctamente.

Un agente que interactúa con el entorno en tiempo real puede construir gradualmente una aproximación de la función *Q(st,at)*.

Antes que comience el aprendizaje, *Q* se inicializa a un valor aleatorio cualquiera. Después, en cada tiempo *t* el agente selecciona una acción *at*, obtiene una recompensa *rt*, introduce un estado nuevo *st* + 1 (Que depende del estado anterior *st* y de la acción seleccionada), y *Q* se actualiza.

Esta actualización sucede cada iteración. Este aprendizaje se modela mediante la función *Q*, a continuación, presentada.

$$Q(s_t, a_t) \leftarrow (1 - \alpha) \cdot Q(s_t, a_t) + \alpha \cdot (r_t + \gamma \cdot maxQ(s_{t+1}, a))$$

donde,

- *rt:* es la recompensa recibida al pasar del estado *st* al estado *st* + 1.
- γ: es el valor de descuento. Este valor aumentará cuando el agente cumpla su tarea (recompensa) y disminuirá, cuando falle su tarea (castigo).
- α: es el "learning rate" o factor de aprendizaje ($0 < \alpha \leq 1$). Es decir, un valor que define el tamaño del paso gradual dentro del algoritmo de optimización de gradiente. Intuitivamente se puede ver como el tamaño del paso de cada iteración mientras se busca que el algoritmo disminuya su error, es decir, qué tan rápido puede ir "aprendiendo". Nota: El valor debe ser óptimo, uno muy pequeño hará el entrenamiento muy tardado, mientras que uno muy grande, podría causar atascos en mínimos locales, provocando así que nunca encuentre el mínimo global de la función de pérdida del error.

Un episodio del algoritmo termina cuando el estado *st* + 1 es un estado final. Para todos los estados finales nunca se actualiza, pero se fija un valor de recompensa para el estado, en la mayoría de los casos puede ser igualado a 0.

Captura la bandera

Captura la bandera es un juego por equipos donde cada uno debe lograr un objetivo, el cual es tomar la bandera en la base del equipo contrario y evitar que su bandera sea tomada. Este juego se ha empleado con ciertas variantes en el mundo de la seguridad informática. Los conocidos juegos CTF (Capture the Flag) o captura la bandera son comunes dentro de la seguridad ofensiva, en el mundo del Ethical Hacking. El objetivo es ingresar a una máquina con algún tipo de vulnerabilidad (o vulnerabilidades) y la labor del jugador es encontrar la vulnerabilidad y explotarla (ejecutarla) para encontrar la bandera (la cual es representada por lo general como un archivo de texto).

Existen propuestas de métodos basados en Q-Learning para que un agente aprenda a resolver tareas de Ethical Hacking como la propuesta por Massimo en su artículo [4]. En este caso vamos a utilizar esto como base para la propuesta del agente malicioso.

El entrenamiento de ambos agentes sucederá definido como un juego de captura la bandera. Donde la meta para el agente malicioso es lograr que el agente defensor deje pasar su código malicioso como si fuese seguro, en este caso, el agente malicioso será recompensado y el agente defensor será castigado.

Por otro lado, el agente defensor debe aprender a ejecutar alguna acción que haga que el malware sea detectado y detenga el ataque.

METODOLOGÍA

En este artículo se propone un nuevo término para el modelo de aprendizaje utilizado, el cual definimos como:

"Aprendizaje por Defensa Reactiva".

Aprendizaje por Defensa Reactiva

Se decidió acuñar este término para hacer referencia a un tipo de aprendizaje automático donde se tienen dos modelos de forma adversaria y con tareas opuestas, uno atacante y uno defensor.

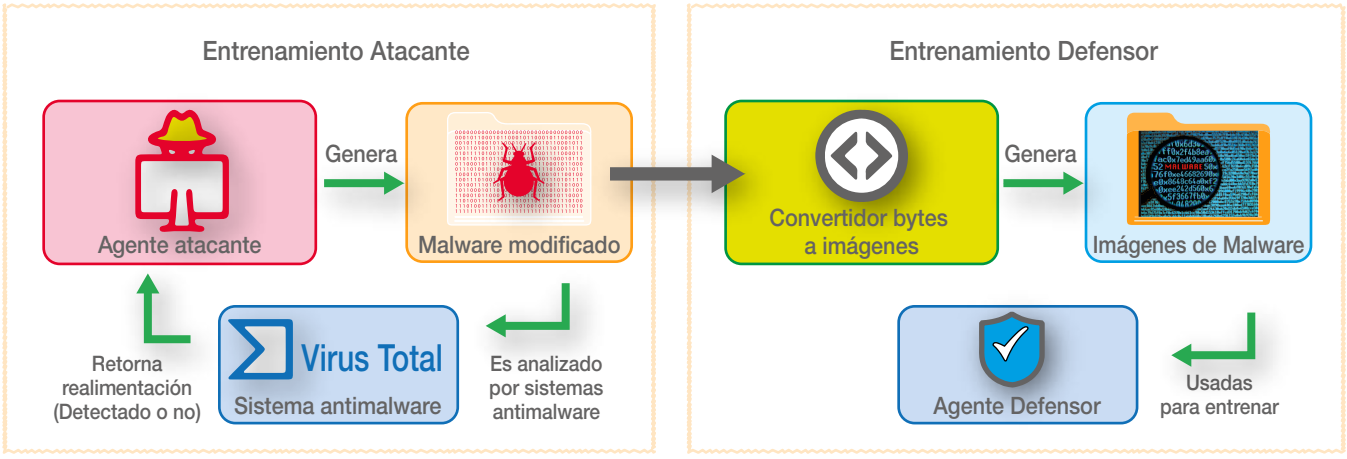
La esencia del término radica en que el modelo defensor aprenderá de las acciones ofensivas del modelo atacante.

A manera de resumen, el método propuesto se basa en dos modelos independientes.

El atacante. Está modelado como un juego de captura de bandera, donde un equipo se encuentra representado por el agente inteligente y el otro equipo se encuentra representado por los motores de antivirus.

El defensor. Aprende del conocimiento generado por el atacante. Se convierte el malware en imágenes y se utiliza para entrenar el modelo defensor.

FIGURA 3. ARQUITECTURA PROPUESTA. APRENDIZAJE POR DEFENSA REACTIVA

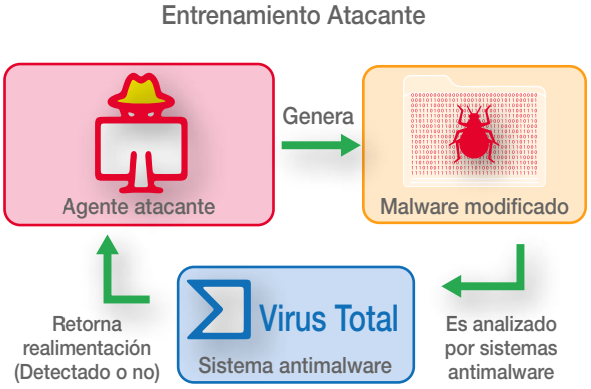


Fuente: Elaboración propia, 2020.

Aprendizaje del Agente Atacante

El atacante se modela como un juego de captura la bandera por equipos, donde el agente inteligente es uno y el antivirus (VirusTotal) el otro equipo. Para efectos prácticos se construye como un modelo de RL donde el agente aprenderá a hacer cambios en el malware a medida que avanzan las iteraciones.

FIGURA 4. ARQUITECTURA DE APRENDIZAJE DEL AGENTE ATACANTE



Fuente: Elaboración propia, 2020.

Arquitectura Atacante

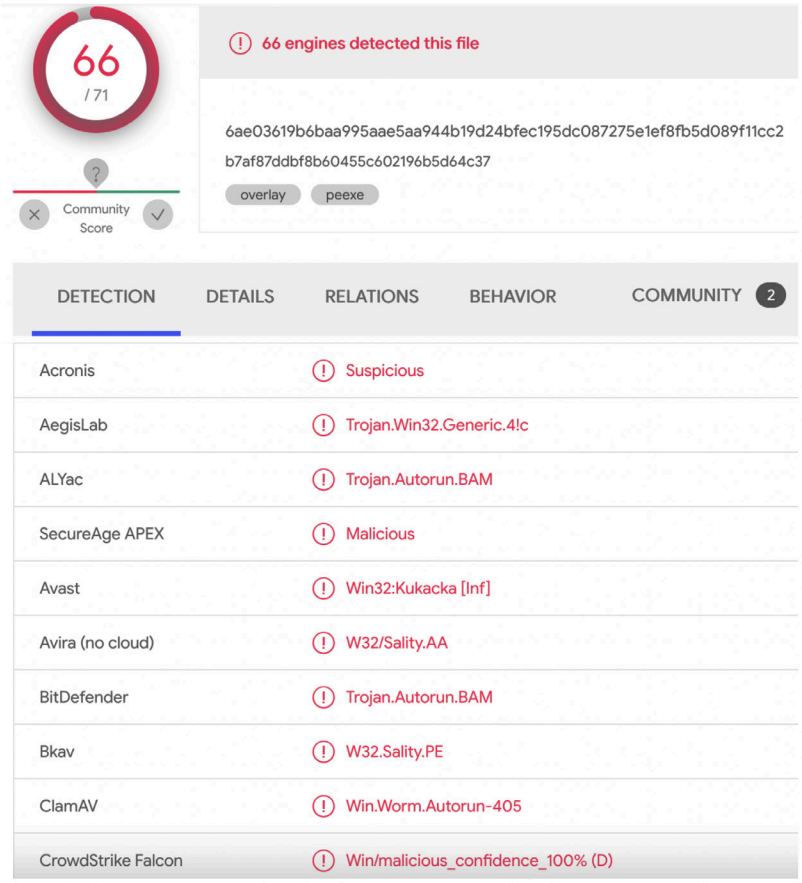
Como se puede apreciar, consiste en:

- **Agente Atacante:** Un agente basado en Q-Learning, que a medida que pasan las iteraciones, va aprendiendo a realizar cambios en los programas ejecutables maliciosos.
- **Malware Modificado:** Esta es una carpeta donde el agente malicioso comienza a guardar las mutaciones o alteraciones que realiza al malware.
- **Sistema Antimalware:** Para este caso se va utilizar VirusTotal como sistema antimalware dado que proporciona una API (Interfaz de programación de aplicaciones) para poder evaluar si el antivirus detectó el malware modificado. Otro aspecto interesante es que VirusTotal cuenta con más de 70 motores de antivirus de diferentes empresas, por lo que se puede evaluar de manera independiente un malware contra cada uno de los más de 70 motores disponibles.

Al modelar este problema como un aprendizaje por refuerzo, se tiene interacción con el entorno (malware y motores de antivirus), lo cual lo hace perfecto para tener realimentación del sistema antimalware y comprobar si realmente está logrando burlar los sistemas de seguridad.

El agente malicioso va a aprender a modificar malware, como si se tratase de un programador malintencionado con gran conocimiento. Es decir, se está entrenando una inteligencia artificial que aprende a modificar código malicioso para hacerlo pasar por código seguro.

FIGURA 5. DEMOSTRACIÓN DE DETECCIÓN DE MALWARE EN VIRUSTOTAL. SE EVIDENCIA COMO UN MALWARE FUE DETECTADO POR 66 MOTORES DISTINTOS, DE 71 DISPONIBLES



Fuente: Elaboración propia, 2020.

Entrenamiento

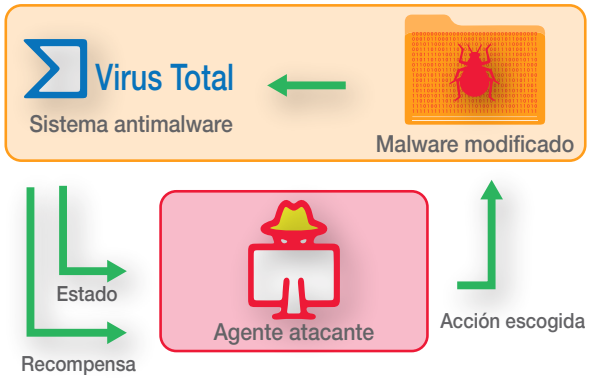
El entrenamiento de este modelo sucede a partir de muestras de malware tomadas de VirusShare [7]. Estas muestras de malware se encuentran como archivos binarios ejecutables o código objeto en formato .bytes.

Al modelar esto como un problema de aprendizaje por refuerzo se obtiene la estructura de aprendizaje (figura 6).

Entorno

El entorno es pieza fundamental en el aprendizaje por refuerzo, pues será quien le dé realimentación al agente. Esto quiere decir que en caso de haber cumplido su tarea será recompensado, mientras que, si falla, será castigado.

FIGURA 6. ESTRUCTURA DE APRENDIZAJE DEL AGENTE MALICIOSO BASADA EN APRENDIZAJE POR REFUERZO. BASADO EN LA ARQUITECTURA PROPUESTA POR ANDERSON EN SU ARTÍCULO [5]



Fuente: Elaboración propia, 2020.

Primeramente, se establece una base de conocimiento y mediante la API de VirusTotal se hace una prueba de cuantos antivirus han logrado detectar la muestra de malware original. Seguidamente el agente escogerá una acción aleatoria dentro su conjunto de acciones disponible para realizar modificaciones al malware.

Estas acciones son las más comunes y utilizadas por los atacantes en vida real.

Dentro de ellas se encuentran:

Crear:

- Crear nuevas secciones de código, que no se utilizarán, pero funcionan para alterar su código (y por lo tanto alterar el hash que permitiría su identificación).
- Crear un nuevo punto de entrada que salte inmediatamente al punto de entrada original o malicioso.

Modificar:

- Importaciones aleatorias.
- Agregar bytes al espacio adicional al final de las secciones.
- Manipular los nombres de las secciones existentes.
- Agregar bytes al final del código PE (código portatil ejecutable).

Agregar:

- Empaquetar o desempacar el archivo usando UPX (empaquetador de ejecutables).

Cuando se haya realizado alguna de estas acciones, el agente enviará de nuevo a VirusTotal su nueva muestra de malware modificada. En caso de mantener el mismo número o mayor de motores que han detectado el malware, será castigado mediante el valor de descuento "γ". En caso contrario, de disminuir el número de motores que han detectado el malware, el agente será recompensado pues habrá realizado bien su tarea haciendo que varios antivirus no hayan detectado su código malicioso.

Inicialmente cometerá muchos errores, por lo que el sistema lo va a castigar. Sin embargo, cuando logre disminuir la cantidad de motores que lo han detectado, este será recompensado y esto hará que el agente vaya comenzado a realizar tareas similares hasta ir encontrando las acciones que mejor recompensa le den a medida que avanzan las iteraciones y va explorando nuevas estrategias.

Este agente tiene dentro de sus opciones 3 grandes grupos de acciones, crear, modificar y agregar.

Estas mutaciones del archivo representan las acciones o movimientos disponibles para el agente dentro del entorno. Estas modificaciones se pueden realizar en un archivo y no rompen el formato ni alteran la ejecución del código.

El agente aprenderá a moverse dentro de este espacio de acciones, buscando lograr colocar el código ejecutable a través del sistema antimalware, es decir, buscando las acciones que hagan que un menor número de motores de antivirus lo detecten.

Estas iteraciones suceden de forma progresiva, el agente debe explorar todas las opciones e ir reconociendo poco a poco las mejores estrategias para lograr burlar la mayor cantidad de antivirus posible.

En resumen, el agente tiene como entrada el estado del entorno (bytes de malware), como salida tiene una acción (modificación de malware) y como realimentación la recompensa o castigo.

Aprendizaje del Agente Defensor

La tarea principal de este modelo es aprender de todas las variaciones generadas por el agente malicioso para así poder detectar código malintencionado de manera autónoma.

Al construir un sistema basado en aprendizaje automático, tiene la capacidad de generalizar, es decir puede predecir para nuevas entradas similares, cuál es la clase a la que mejor se adapta.

Al tener la capacidad de generalizar, las variaciones de un atacante al código no deben ser precisamente las mismas, dado que puede reconocer similitudes entre ellas. Generando así, una gran ventaja sobre un sistema tradicional de firma de código.

Arquitectura

La arquitectura se constituye de las siguientes partes.

- **Convertidor de bytes a imágenes:** Este es un método propuesto por Nataraj [2] donde propone convertir un archivo .bytes a una imagen para poder clasificarla. En este artículo se utiliza esto como estado del arte para construir sobre él.

FIGURA 7. ARQUITECTURA DEL MODELO DE DEFENSA

Fuente: Elaboración propia, 2020.

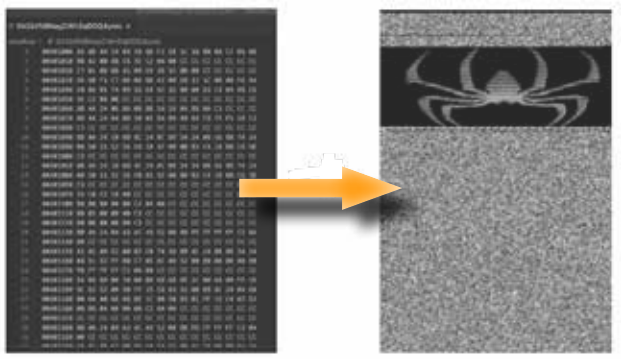
Básicamente se utiliza el siguiente método:

FIGURA 8. ARQUITECTURA DE CONVERSIÓN DE ARCHIVO MALICIOSO EN IMAGEN

Fuente: Elaboración propia, 2020.

En pocas palabras, toma un archivo binario como el que se muestra a la izquierda en la figura 9 y genera una imagen como la que se muestra en la parte derecha de la figura 9.

FIGURA 9. MUESTRA DE CONVERSIÓN DE ARCHIVOS. A LA IZQUIERDA, UN MALWARE EN FORMATO BYTES. EN LA PARTE DERECHA, EL MISMO MALWARE CONVERTIDO EN IMAGEN



Fuente: Elaboración propia, 2020.

- **Carpeta con imágenes de malware:** Es una carpeta que contiene todas las muestras de malware obtenidos de VirusShare [7], convertidas en imágenes.
- **Agente Defensor:** Este es un agente basado en Deep Learning. Toma como entrada las imágenes de malware modificado, que han logrado evadir varios motores de antivirus, y se entrenará con ellas para aprender a identificar tanto el malware, como la ofuscación o los métodos que el agente malicioso haya utilizado para burlar los sistemas antivirus. Esto corresponde al proceso de **Aprendizaje por Defensa Reactiva.**

La manera en la que aprende el modelo defensor es más sencilla de comprender que la del modelo del atacante, pues el defensor no tiene realimentación de un entorno, como lo tenía el atacante.

Entrenamiento

El entrenamiento se realiza de forma supervisada en donde las muestras de malware están clasificadas por 4 tipos de malware.

- **Backdoor:** Código malicioso que permite acceso y control remoto de un equipo.
- **Ransomware:** Es un código malicioso cuyo fin es restringir el acceso a un sistema o determinada parte de un sistema, haciendo uso de criptografía.
- **Troyano:** Es un programa que aparenta ser inofensivo, sin embargo, su fin real es malicioso.
- **Worm:** Es un malware capaz de moverse por sí solo entre varios equipos dentro una red y no depende de la ejecución de algún otro programa.

Estos malwares se encuentran organizados por carpetas dentro de la carpeta de malware que, al momento de convertir los archivos ejecutables a imágenes, mantiene el formato de organización, de este modo se sabe cuál imagen pertenece a cuál tipo de malware. Gracias a esta supervisión el modelo es capaz de aprender las diferencias entre los tipos de malware y detectarlos correctamente.

Modelo de Deep Learning

El modelo es una Red Neuronal Convolucional, construida como se muestra a continuación:

FIGURA 10. RESUMEN DEL MODELO CNN UTILIZADO

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_14 (Conv2D)           (None, 63, 63, 30)        390
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 15, 15, 30)        0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 12, 12, 15)        7215
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 6, 6, 15)          0
_________________________________________________________________
dropout_14 (Dropout)         (None, 6, 6, 15)          0
_________________________________________________________________
flatten_7 (Flatten)          (None, 540)               0
_________________________________________________________________
dense_21 (Dense)             (None, 128)               69248
_________________________________________________________________
dropout_15 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_22 (Dense)             (None, 60)                7740
_________________________________________________________________
dense_23 (Dense)             (None, 30)                1830
_________________________________________________________________
dense_24 (Dense)             (None, 25)                775
=================================================================
Total params: 87,198
Trainable params: 87,198
Non-trainable params: 0
_________________________________________________________________
```

Fuente: Elaboración propia, 2020.

En resumen, el modelo es una arquitectura de deconvolución, espera imágenes de 64 píxeles x 64 píxeles y progresivamente va reduciendo su tamaño para extraer los patrones hasta llegar a las capas planas y densas donde en la última capa utiliza una función softmax para determinar probabilidad de pertenencia a cada una de las 4 clases (Backdoor, Ransomware, Troyano, Worm).

RESULTADOS

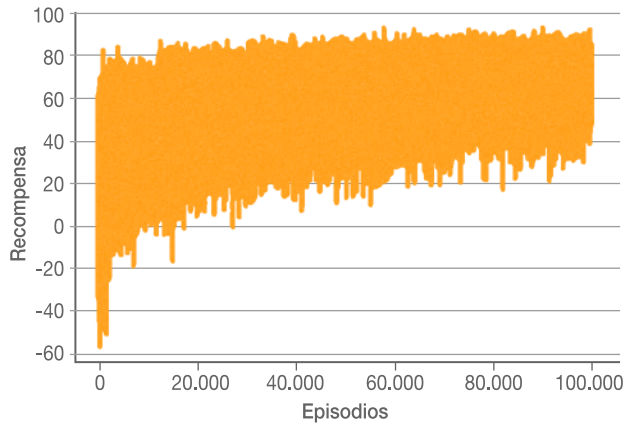
Al concluir exitosamente los entrenamientos de ambos modelos se va a estudiar los resultados de ambos modelos de forma independiente.

Entrenamiento del agente atacante

Este entrenamiento se realizó basado en un número de 100.000 épocas o iteraciones del modelo. Podemos apreciar en el gráfico como el Reward o recompensa (factor de premio dado cada vez que logra burlar uno o más motores de antivirus) va aumentando a lo largo de los episodios de entrenamiento.

Esto confirma que el agente, en efecto, ha aprendido a realizar su tarea correctamente.

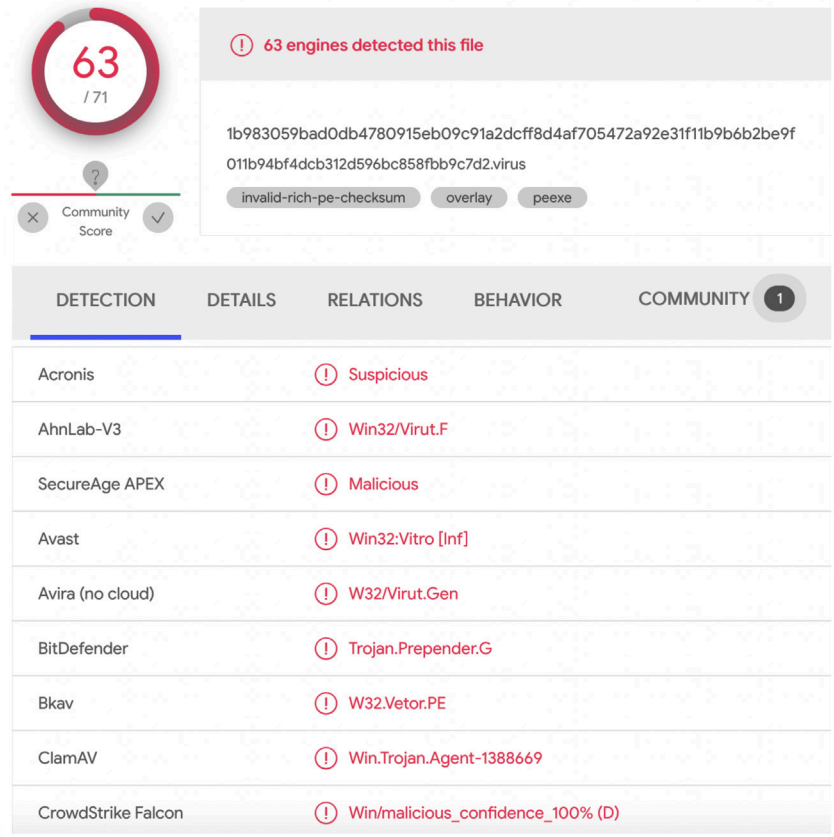
FIGURA 11. RECOMPENSA A LO LARGO DEL ENTRENAMIENTO



Fuente: Elaboración propia, 2020.

A continuación, se muestra un ejemplo donde para el mismo archivo malicioso, inicialmente fue detectado por 63 motores.

FIGURA 12. MOTORES DE ANTIVIRUS QUE DETECTARON EL MALWARE ORIGINAL



Fuente: Elaboración propia, 2020.

Después de que el agente realizó cambios en el archivo, podemos ver que fue detectado por 43 motores. Es decir, logró hacer que 20 motores de antivirus, dejaran pasar el malware, como si fuera un archivo seguro (figura 13).

Como se evidencia, el agente malicioso ha logrado aprender a burlar antivirus con éxito.

Entrenamiento del agente defensor

La validación del modelo se realiza utilizando validación cruzada, específicamente el algoritmo K-Folds. La validación cruzada es un procedimiento de remuestreo que se utiliza para evaluar modelos de aprendizaje automático en una muestra de datos limitada. Es decir, los datos totales se convierten en varios subconjuntos.

Se realizaron 10 Folds o subconjuntos por iteración, cada una consta de 10 episodios.

Como media aritmética de todos los Folds, se obtuvo ***una precisión del 92% para la detección y clasificación del malware*** en sus 4 clases (Backdoor, Ransomware, Troyano, Worm).

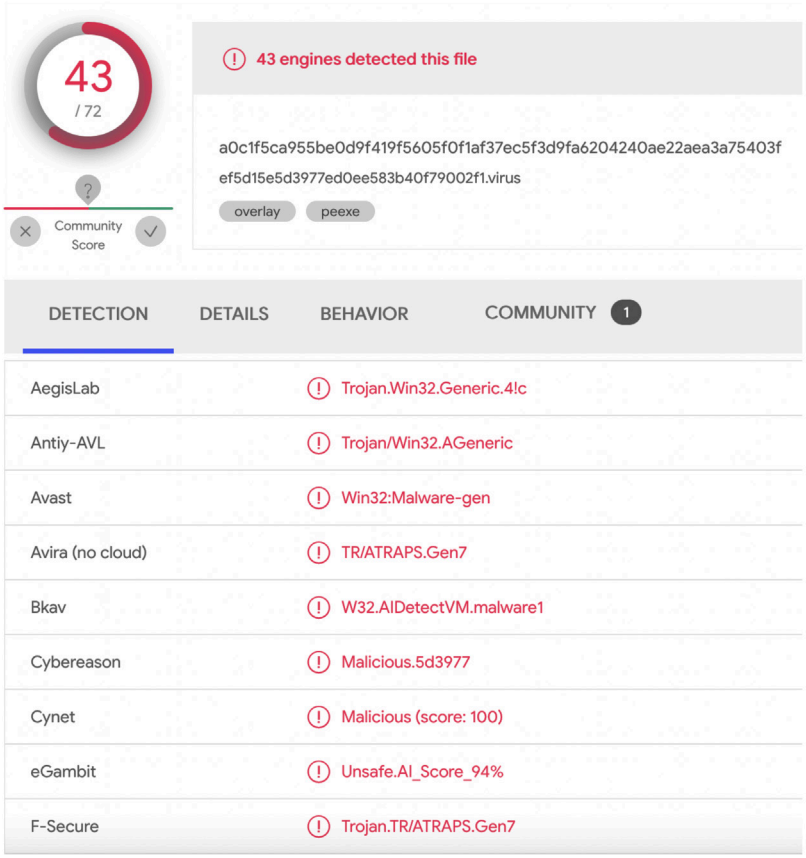
De esta manera se ha logrado que el algoritmo aprenda a identificar los patrones de malware y los patrones generados por las modificaciones más utilizadas por los atacantes.

Esto confirma que nuestra propuesta posee una clara ventaja para detectar rápidamente cualquier intento de burlar un sistema antimalware.

CONCLUSIONES

Al concluir esta esta investigación podemos mostrar varios hallazgos que vamos a ir detallando a continuación.

FIGURA 13. CANTIDAD DE MOTORES QUE DETECTARON EL MALWARE MODIFICADO



Fuente: Elaboración propia, 2020.

"El atacante siempre tendrá la ventaja"

Durante el diseño de este experimento se ha reafirmado que existe un problema dentro del mundo de la ciberdefensa y es que siempre el atacante (ya sea un agente como en este artículo, o uno de la vida real), siempre tendrá la ventaja, pues puede decidir libremente qué acción ejecutar, mientras que la parte defensiva, solamente puede aprender de ataques pasados y tratar de predecir nuevos ataques, pero no tendrá certeza de ello pues esto implica un proceso de aleatoriedad, ya que el atacante puede ser muy creativo implementado nuevos métodos nunca antes vistos, dejando así perdido al defensor.

Una analogía de esto son los penales en fútbol, el portero puede tratar de predecir hacia dónde irá la bola, pero no tendrá certeza de ello, mientras que el tirador puede improvisar y tomar una acción aleatoria. Lo mismo ocurre con la ciberseguridad.

"Herramienta para analizar las diversas casas de motores de sistemas antimalware"

Una posible aplicación en la industria del trabajo realizado en este artículo es que representa una manera fiable de elegir una casa fabricante de un sistema antimalware.

Esto porque el modelo propuesto permite una simulación real de posibles escenarios de ataques que se podrían generar debido a las modificaciones reales en el archivo malicioso (tal y como lo haría un atacante en la vida real).

De esta manera las empresas pueden utilizar el agente entrenado para generar variaciones a códigos maliciosos y evaluar cuáles motores tienen una mejor capacidad de detección a pesar de todas las mutaciones o modificaciones que se le hagan a los archivos maliciosos, lo que representa una excelente forma de evaluar cual motor de antivirus comercial utilizar.

"Herramienta de generación de nuevos hashes"

Otro aporte importante en la industria es que ***permite la generación automática de nuevos hashes para los motores basados en detección por firma de código*** lo que permitiría mejorar exponencialmente los rangos de detección de sus motores.

"Liviano y rápido"

Otra ventaja es que el modelo defensor puede ser exportado, por lo tanto, cuando se quiera analizar un nuevo programa para detectar si es malicioso, nuestro motor sólo necesita convertir el código a imagen y hacer una predicción sobre la misma y lo mejor de todo es que esto ocurre ¡En menos de un segundo!.

Tampoco necesita tener una base de datos conectada de manera local, ni remota, por lo que el tiempo de detección se reduce considerablemente, esto gracias a que no debe calcular hashes.

Tampoco existe tiempo de latencia entre una base de datos remota ni comparaciones entre todos los hashes de una base de datos. Lo cual supone un método de detección más rápido y liviano que un motor basado en firmas de código.

"Mayor rango de detección"

El diseño aquí propuesto, como se ha evidenciado, propone una solución eficaz en cuanto a la detección de malware con variaciones generadas por un atacante que además es ***capaz de aprender los patrones de las técnicas más habituales para burlar estos sistemas***.

La mayor parte de los atacantes son personas con un conocimiento en seguridad que utilizan herramientas preexistentes para realizar estos cambios, por ello, de forma muy sencilla el modelo aquí propuesto podrá identificar estos patrones de cada herramienta, convirtiéndolas en inútiles dado que el modelo habrá aprendido a identificar cómo opera y cuando un Script Kiddie (Término utilizado para describir atacantes con poco conocimiento, que hacen uso de herramientas existentes) utilice una de estas herramientas y por lo tanto podría ser detectada con facilidad.

En caso de atacantes más experimentados (que hacen estas modificaciones de forma manual), el modelo es capaz de generalizar patrones similares, por lo que algunas técnicas sofisticadas podrían ser detectadas.

Por supuesto entre más conocimiento tenga el modelo, mejor rango de detección tendrá (como cualquier sistema basado en inteligencia artificial debe ser re-entrenado constantemente para poder mejorar su precisión).

LIMITACIONES

Los agentes modelados con aprendizaje por refuerzo logran ser realmente eficientes al cabo de muchas iteraciones. Esto es normal en algoritmos evolutivos, (de ahí surge el nombre), pues se deben dejar entrenar un período de tiempo considerable, dado que a medida que avanza el tiempo, los agentes aprenden nuevas estrategias. Esto representa la primera limitación para el modelo atacante. Dado que se debe dejar un número óptimo de horas que sea suficiente para que los agentes puedan aprender a ser más creativos. Es decir, continuar mejorando su recompensa. En la figura 11 pudimos apreciar como el modelo tiene variaciones importantes en la recompensa, no es estable, idealmente se busca una estabilidad en esto, sin embargo, esto es causado debido a que el agente probará acciones aleatorias y algunas serán mejores que otras (el aprendizaje no es perfectamente lineal), es decir que a medida que avanzan las iteraciones, irá aprendiendo, pero también probando y cometiendo errores.

"Nuestro modelo aprende de manera autónoma por medio de prueba y error lo que hace que necesite tiempo para encontrar la mejor solución".

La segunda limitante que tienen estos modelos es que si se recuerda la formulación de un problema de aprendizaje por refuerzo se tiene $\langle S,A,T,R \rangle$ donde A es un conjunto de acciones. El problema radica en que los agentes aprenderán a hacer combinaciones o escoger la acción óptima según sea el caso, sin embargo, se limita a exclusivamente las acciones contenidas en ese conjunto. De modo que el agente no podrá conocer cualquier otra acción no definida dentro del conjunto, es decir que no sabe que puede lograr utilizar nuevas acciones que no están definidas dentro de ese grupo por lo tanto se limita a escoger dentro de las acciones que tiene.

Cualquier acción que no se encuentre contenida en este grupo, será obviada. Lo cual supone, una limitante importante en estos modelos de inteligencia artificial.

Esta limitante se encuentra dada por la definición matemática del algoritmo, sin embargo, la mayoría de modelos de inteligencia artificial poseen esta limitante, pues es muy difícil lograr modelar un grupo de acciones de todo el universo existente, por lo que por lo general se utiliza teoría de conjuntos para modelar los algoritmos y el programador debe agregar de manera manual las posibles acciones al conjunto.

¿QUÉ SIGUE?

El presente trabajo supone una base técnica para futuros trabajos relacionados. Por supuesto existe un gran campo de implementaciones para mejorar el método aquí propuesto.

El primer aspecto para tener en cuenta es el número de horas de entrenamiento del modelo atacante, pues como se mencionó anteriormente, estos modelos a medida que avanza el tiempo y las iteraciones van aumentando, los agentes aprenden a mejorar sus estrategias y se vuelven mejores resolviendo sus tareas. Por esta razón se sugiere entrenar al agente un número importante de horas esperando nuevas acciones o estrategias encontradas por los agentes.

También es posible definir de alguna otra forma el juego como tal, al final se recuerda que esta metodología propuesta entrena los agentes de forma independiente. Por lo que se podrían hacer pruebas con agentes modelados de forma conjunta y con un espacio de acciones mucho mayor.

Algo interesante que se podría lograr hacer es que el agente defensor tome acciones más específicas, de acuerdo con el tipo de ataque que detecte. Es decir, si el agente logra identificar que un ataque provino de un vector particular como algún puerto abierto, pueda tomar la decisión de cerrarlo. Esto se puede lograr modelando el defensor como agente de RL y dejando que el agente experimente por sí solo y aprenda cuándo es conveniente tomar una acción como esta.

Otra recomendación para la arquitectura actual sería aumentar la cantidad de tipos de malware detectado, pues de momento el experimento utilizado en este artículo, cuenta únicamente con 4 tipos de malware.

Para mejorar la precisión y agrandar la cantidad de tipos de malware que se pueden detectar es necesario que el modelo atacante cuente con más muestras de malware en formato .bytes de más clases (entre más ejemplos de cada clase se tengan, más preciso será el modelo de detección).

Por último, otro trabajo futuro recomendado es realizar una **prueba 10-10-10** (10 muestras de malware, evaluadas 10 veces, para obtener los mejores 10 motores antimalware). De esta forma, se podría evaluar los distintos motores de antimalware disponibles y obtener un top 10 o los 10 motores que poseen una mejor capacidad de detección de malware, incluso con variantes en su código. De modo que se tenga un parámetro real, para poder tomar la decisión de cuál utilizar.

BIBLIOGRAFÍA

[1] Saxe, J. y Berlin, K. (2015). *Deep neural network based malware detection using two dimensional binary program features. InMalicious and Unwanted Software (MALWARE).* 2015 10th International Conference on, pp. 11-20. IEEE.

[2] Nataraj, L.; Karthikeyan, S.; Jacob, G. y Manjunath, B. (2011). *Malware Images: Visualization and Automatic Classification*. 10.1145/2016904.2016908.

[3] Silver, D., Huang, A., Maddison, C. *et al.* (2016). Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 484-489: https://doi.org/10.1038/nature16961

[4] Massimo Zennaro, F. y Erdodi, L. (2020). *Modeling Penetration Testing with Reinforcement Learning Using Capture-the-Flag Challenges and Tabular Q-Learning.* Recuperado de: http://arxiv.org/abs/2005.12632 arXiv:2005.12632

[5] Anderson, H.Y.; Kharkar, A.; Filar, B.; Evans, D. y Roth, P. (2018). *Learning to Evade Static PE Machine Learning Malware Models via Reinforcement Learning.* Recuperado: http://arxiv.org/abs/1801.08917 arXiv:1801.08917

[6] Thanh Thi, N. y Vijay Janapa, R. (2019). Deep Reinforcement Learning for Cyber Security. Recuperado de: http://arxiv.org/abs/1906.05799 arXiv:1906.05799

[7] VirusTotal.com. *(s.f.).* Recuperado de: https://www.virustotal.com/gui/