ESPECIES INDICADORAS ARTIFICIALES: UN CLASIFICADOR ADAPTATIVO BASADO EN VIDA ARTIFICIAL

por

Matías Octavio Lermanda Sandoval

Patrocinante: María Angélica Pinninghoff

Memoria presentada
para la obtención del título de

INGENIERO CIVIL INFORMÁTICO

Departamento de Ingenería Informática y Ciencias de la Computación

de la

UNIVERSIDAD DE CONCEPCIÓN

Concepción, Chile
Marzo, 2018

Urbi et Orbi

Resumen

El problema de la seguridad en la informática es algo que se ha tratado de resolver de diversas maneras a lo largo de los años, en particular con la llegada de internet, que ha permitido que salgan a la luz muchas formas de ataque. Por ello es importante siempre estar a la vanguardia en lo que a defensa se refiere. Para esta memoria se trabajó sobre la base de un método de especies indicadoras artificiales ya existente, buscando mejoras a éste. Se realizaron pruebas de las mejoras encontradas y se demostró que el método mejorado es más efectivo a la hora de determinar si hay un ataque en curso. Para finalizar, se concluyen diversas posibilidades de trabajo futuro, incluyendo algunas mejoras propuestas y una implementación en un lenguaje que permita una aplicación en el mundo real.

Índice general

Índice de figuras

Capítulo 1

Introducción

En informática, la seguridad es un tema de suma importancia, cada mes se descubren cientos de formas nuevas de atacar distintos tipos de sistemas [6], lo que ocasiona que cualquier intento de detectar un ataque se vuelva obsoleto con relativa rapidez.

La cantidad de ataques de denegación de servicio con un ancho de banda superior a 1 Gbps aumentó en un 172 por ciento en 2016 y se proyecta que aumente 2.5 veces para un total de 3.1 millones de ataques para el año 2021 [4]. Por estos motivos es que las herramientas existentes deben mantenerse constantemente actualizadas para tener al día sus capacidades de protección del sistema ya que al verse enfrentadas a un nuevo tipo de ataque, si no es reconocido como una amenaza, generalmente es ignorado.

Una especie indicadora es aquella que puede definir una característica de su medio ambiente [9], por ejemplo, la población de una cierta especie puede indicar la disponibilidad de recursos o algún cambio en el medio. La idea de una Especie Indicadora Artificial (EIA), es tomar este principio y aplicarlo con el fin de obtener un clasificador de propósito general basado en una población de agentes sensibles a cambios en el ambiente, este ambiente corresponde al sistema que quiere ser monitoreado, en este caso, la actividad de una red de computadores.

Los agentes de este sistema, a través de diversas características como población y energía disponible, muestran señales que indican que alguna situación se desvía de la normalidad. La idea principal es aplicar los conceptos de genética y adaptación a un contexto de seguridad en redes de computadores. Bajo esta premisa, los cambios en la población permitirían detectar anomalías en una red, esto sería un indicador clave de que la red está pasando por alguna situación que se sale de la normalidad.

El sistema fue implementado sobre la plataforma de programación de multiagentes NetLogo y posee la capacidad de detectar un cierto número de anomalías indicadoras

de posibles ataques a la red mediante la comparación con la tendencia de ciertos parámetros tales como la población de agentes, energía promedio de los agentes y la cantidad de partículas indicadoras en el ambiente, estas últimas se corresponden con la cantidad de características buscadas en los paquetes, que fueron efectivamente encontradas.

Para esta memoria, se propone lograr una implementación similar y mejorar la realizada en trabajos relacionados [2, 18], es decir, crear un sistema base y, a partir de eso, investigar e implementar una serie de actualizaciones que permitan mejorar el desempeño del sistema (clasificador), estas mejoras se pueden cuantificar utilizando las medidas de desempeño: precisión, sensitividad, exactitud y especificidad.

La comparación con el sistema de base será realizada con capturas de tráfico de una red normal, es decir, una red que no es objeto de ataque y/o comportamiento sospechoso y con capturas de tráfico de una red bajo ataque, los tipos de ataque seleccionado fueron los siguientes:

- DoS (Denial of Service): Ataque de denegación de servicio, se caracteriza por tener el objetivo de saturar la capacidad de una cierta máquina o sistema por cualquier medio disponible.

- Probe: Este no representa un ataque propiamente tal, es una fase inicial previa a la realización de cualquier intento de disrupción del servicio, sirve para recopilar información acerca de una máquina o máquinas objetivo.

- MitM (Man in the Middle): Ataque de "Hombre en el Medio", esta forma de ataque se utiliza para engañar a otras máquinas y lograr que los paquetes que ellas envíen pasen primero por la máquina atacante, generalmente se logra convenciendo a las otras máquinas de que la máquina atacante es el router.

Los objetivos generales de este trabajo son, en primer lugar, lograr la implementación del modelado de un sistema de Especies Indicadoras Artificiales en NetLogo, realizar pruebas del modelo en un contexto de seguridad en redes, investigar e implementar posibles mejoras al modelo y finalmente evaluar y comparar los resultados obtenidos con las estadísticas que entrega el sistema de base.

El sistema funciona como un IDS (Intrusion Detection System), en particular como un NIDS (Network Intrusion Detection System), es decir, analiza una red y reporta cualquier actividad sospechosa, opera sobre un modelo de detección basada en anomalías, esto quiere decir que la manera en que determina si cierto tráfico es malicioso es por una desviación de parámetros que el sistema considera como normales, esta "normalidad" se determina por un cierto período de adaptación inicial de la población de agentes a las condiciones normales del entorno de la red.

1.1. Objetivos

1.1.1. Objetivos Generales

- Implementar el modelado del sistema de Especies Indicadoras Artificiales en NetLogo.
- Realizar pruebas del modelo en el contexto de seguridad en redes.
- Investigar e implementar posibles mejoras al modelo.

1.1.2. Objetivos Específicos

- Identificar las variables de la población importantes para medir cambios en el ambiente.
- Identificar los parámetros relevantes de los paquetes de datos, que permitan modelar de manera efectiva el medioambiente de la red.
- Evaluar la relevancia de las variables y parámetros elegidos con respecto a la seguridad de una red de comunicación.
- Descubrir posibles simplificaciones del modelo.
- Evaluar y comparar los resultados con el modelo pre-existente del sistema.

Capítulo 2

Trabajo Relacionado

El problema de la detección de intrusos ha sido enfrentado desde varios ángulos [3], la forma más común de detección es la llamada S*ignature-based detection* (Detección por patrón), este método utiliza configuraciones conocidas de paquetes maliciosos, o un modelo de ellas que se comparan con las condiciones de la red, esto hace que el sistema sea incapaz de aprender, ya que estas definiciones vienen configuradas por defecto y/o deben ser agregadas de forma manual, en otras palabras, representan un conjunto de reglas estáticas.

Entre las técnicas que utilizan esta forma de detección encontramos métodos de reconocimiento de patrones, tales como el observado por el software SNORT [14], un ejemplo de NIDS que utiliza patrones para reconocer ataques, la técnica de reconocimiento de patrones se enfoca en reconocer patrones y regularidades en los datos. Generalmente se entrena a partir de conjuntos de datos etiquetados pero también puede ser objeto de entrenamiento no supervisado. Los algoritmos de reconocimiento de patrones generalmente tratan de entregar una respuesta para cada entrada pero también pueden hacer supuestos, si el patrón no calza perfectamente con ninguno conocido.

Otra técnica es la de reglas de implicación. Esta técnica utiliza un conjunto de reglas predefinidas que al relacionarse con eventos de la red, permiten inferir que hay un ataque en progreso, un ejemplo de sistema que utiliza esto es P-BEST [21].

Una tercera técnica de detección es por medio de *Data mining*, en este caso, la idea es eliminar la necesidad de generar manualmente un modelo de tráfico ya que este se construye automáticamente a partir de un conjunto de datos de referencia, es decir, el sistema se encarga, por sí solo, de generar un modelo que le permita una correcta identificación de un ataque, una ventaja de este método es que no sólo es capaz de detectar los ataques a partir de los cuales se generaron los modelos, sino que

también puede detectar algunas variaciones de estos

Las técnicas de *Machine learning* también son utilizadas, estas incluyen técnicas como, redes neuronales [5] y programación genética [13], entre otros. La principal desventaja de estos métodos es que requieren un tiempo de entrenamiento variable y un conjunto de datos etiquetados que les permitan aprender a clasificar de manera correcta las condiciones del medio, además, si se ven enfrentados a una situación nueva, son incapaces de adaptarse, debido a que su entrenamiento es exclusivamente en la fase inicial.

La segunda forma para detectar ataques es conocida como *Anomaly-based detection* (Detección basada en anomalías), esta opera casi de manera opuesta a la anterior, aquí se busca crear un perfil de tráfico normal, por lo tanto, todo tráfico que se desvíe de esta normalidad puede ser considerado como un ataque, esta metodología tiene la ventaja de que es sensible a cambios en el ambiente y por lo tanto cualquier ataque que deje una marca puede ser detectado. En el caso de que un ataque no produzca ningún efecto perceptible en la red, entonces no podría ser detectado por este método.

Una desventaja importante de la detección basada en anomalías es que puede tener un alto porcentaje de falsos positivos, esto se debe a que a veces un cambio menor puede provocar una respuesta de alarma por parte del sistema, puede depender además de qué es lo que se está observando exactamente en la red.

Algunos métodos que utilizan esta forma son algunos modelos estadísticos. Estos mantienen un par de perfiles de la red al mismo tiempo, uno para lo que está ocurriendo actualmente en la red y uno que se observó previamente, a medida que las características de la red cambian, se actualiza el modelo actual, este es luego comparado con el modelo de base y se dispara una alerta si difieren en cierta medida. Los perfiles generalmente se basan en la observación de ciertos parámetros que fueron determinados como relevantes previamente, como ejemplo de software que utiliza este método tenemos a SPICE [19] (*Stealthy Port scan and Intrusion Correlation Engine*), su arquitectura consiste de un sensor de anomalías y un componente encargado de hacer correlaciones, el sensor monitorea la red y asigna un puntaje de anomalía a cada evento, mientras más veces sea observado un cierto paquete, mayor será su puntaje de anomalía, llegado un cierto nivel y se disparan las alarmas.

También se puede usar *Machine learning* con esta forma, uno de los intentos en esta área es el sistema PHAD [17] (*Packet Header Anomaly Detection*), este realiza detección de anomalías mediante el análisis de las cabeceras de los paquetes, considera algunas características de los campos de Ethernet, IP y de la capa de transporte, en este caso el puntaje de anomalía de estos se calcula en el momento y sin tener en cuenta eventos pasados.

Otro sistema en la misma área es el ALAD [16] (Application Layer Anomaly Detection), en este caso, en vez de considerar paquetes individuales, se revisan las conexiones TCP (*Transmission Control Protocol*) entrantes, al mismo tiempo que se observa el contenido del paquete.

El *Data mining* también puede ser utilizado para la detección de anomalías, se ha propuesto el uso de algoritmos que generen reglas inductivas [10]. Estos algoritmos combinan patrones de asociación con patrones para clasificar el tráfico, considera características de bajo y alto nivel de los paquetes, con esto se genera un modelo a partir de tráfico que no es objeto de ataque y este modelo se compara con el tráfico actual para determinar si hay una anomalía.

El uso de Especies Indicadoras Artificiales (EIA) para resolver este problema [2, 18] corresponde a una variante de la utilización de la detección basada en anomalías, este método se aprovecha de la adaptación de sus agentes para crear un sistema que se adapta progresivamente, esto causa modificaciones continuas en la población de agentes mediante la selección natural y mutaciones aleatorias.

En esta técnica, se utiliza una población de agentes que se alimentan de diferentes partículas existentes en el ambiente, es decir, los agentes interactúan con estas partículas y reaccionan de acuerdo al tipo de partícula y al tipo de agente. Las partículas son creadas en respuesta a diversas condiciones del medio y representan ciertas características de la red que se desea observar. Los agentes son gobernados por un conjunto de reglas que determinan su comportamiento y mediante la medición de diversas características del modelo, se puede determinar si hay un ataque en progreso.

Para determinar las características de la red se observan diversos elementos de los paquetes, entre estos se cuentan el servicio al cual corresponde el paquete, la función de este (SYN, ACK, FIN, RST), entre otros. Cada una de estas características esperadas

genera una partícula de cierto tipo, por otro lado, los agentes tienen un genoma que le indica a éste si una cierta partícula le aporta o le quita energía. Cuando una partícula colisiona con un agente, se compara el tipo de partícula con el genoma del agente, éste le indica el valor nutricional de cada alimento que consume, pudiendo tener un valor positivo o negativo. El valor nutricional ocasiona que la energía del agente cambie, al llegar a cierto nivel de energía el agente se reproduce de manera asexual, esto produce un hijo con un código genético igual al del padre exceptuando mutaciones que corresponden a una variación aleatoria que se hace del genoma del padre.

La forma en que el sistema clasifica el estado de la red es por medio de la reacción a ciertas características, si estas características se desvían de ciertos límites predeterminados, entonces se considera que hay un ataque en progreso.

Una desventaja de este método es que el sistema requiere de un tiempo inicial de adaptación, este tiempo puede ser variable y depende además de la implementación que se haga, esta desventaja implica que si un ataque ocurre durante este período de adaptación al ambiente, las consecuencias podrían ser imprevisibles, es decir, podría darse el caso de que un ataque no sea reconocido como tal o que la condición normal sea clasificada como ataque.

En un ambiente que sea extremadamente cambiante, puede darse la situación de que a la EIA le sea muy difícil adaptarse, en este caso, sería necesario reajustar los parámetros para poder tener alguna posibilidad de éxito en la clasificación.

Entre las ventajas de las EIA se cuenta que no necesitan de un período de entrenamiento, ya que se adaptan de manera continua al ambiente de la red, esto significa que una vez determinados los parámetros con los cuales el sistema opera de manera efectiva y eficiente, no es necesario hacerle ningún ajuste manual, es decir, el sistema se adapta sin intervención de una persona a las condiciones cambiantes del ambiente.

Capítulo 3

Sistema Desarrollado

3.1. Modelo

El sistema fue creado con la herramienta para programación de multiagentes Netlogo 3.1 , el modelo se describe como un mundo bidimensional en el cual conviven dos especies, estas son:

- Agentes
- Partículas

Los agentes son entidades estáticas creadas con un código genético con valores aleatorios, este código genético es el que permite que, a través de ciertos procesos que serán explicados más adelante, sobrevivan sólo los más fuertes y que estos sean los que generen descendencia. Cada agente tiene algunos datos clave asociados a él, estos son:

- Energía (E_x): Indica la cantidad de energía que almacena el agente, esta energía es utilizada para mantener su metabolismo funcionando y para reproducirse.

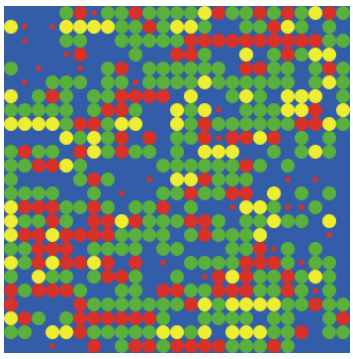

Figura 3.1: Ejemplo de una vista del mundo

- Código genético (Θ): Determina las aptitudes del agente, en otras palabras, su capacidad para sobrevivir a alguna determinada característica del ambiente. $\Theta = [1, 2, 3, ..., n]$, donde cada $i \in \Theta$ representa un gen del código y con n la cantidad de características buscadas en cada paquete.

Los agentes requieren de un cierto nivel de energía para poder reproducirse, llamado Energía de Reproducción (E_r), la cuál se determina empíricamente, éstos se reproducen de manera asexual, actuando sobre su código genético un operador de mutación que corresponde a una permutación aleatoria de dos elementos de su código genético Θ, esta pequeña variación en el hijo determina la afinidad de este por el ambiente.

Los agentes son gobernados por un conjunto de reglas enunciadas a continuación:

1. $E_x > E_r \wedge espacio(x) \implies crearNuevoAgente()$
2. $E_x \leq 0 \implies agenteMuere(x)$
3. $E_x = E_x - m$
4. $colision(x, \rho) \implies comer(x, \rho)$

Estas reglas se explican de la siguiente manera:

1. Si un agente contiene más energía que la requerida para reproducirse y hay espacio disponible a su alrededor, entonces debe crear un hijo, a efectos prácticos, la restricción de espacio significa que en la vecindad de Von Neumann del agente, debe haber al menos un espacio que no contenga a ningún otro agente, se puede dar el caso de que 2 agentes consideren como vacío un cierto espacio y creen un hijo al mismo tiempo, tal que ambos hijos ocupen el mismo espacio, en este caso se elimina uno de los hijos que comparten espacio de manera aleatoria.
2. Si un agente agota toda su energía, este muere, esto puede ocurrir principalmente por una de dos razones: puede ser que pierda energía por la que gasta en mantener su metabolismo, o puede recibir una partícula para la cual no está adaptado, dando como resultado, al consumirla como alimento, que esta sea

considerada como veneno por el agente y provoque una pérdida de energía en el mismo; usualmente la pérdida de energía ocurre por una combinación de ambos.

3. En cada tick o paso de la simulación, los agentes pierden una determinada cantidad m de energía, esta energía está destinada a mantener con vida al agente, o en otras palabras, esto simula su metabolismo.

4. Si un agente x colisiona con una partícula ρ, el agente obtiene energía de esta de acuerdo con la ecuación de la figura 3.2. Este proceso no elimina a la partícula y tampoco le resta una cantidad de energía proporcional a la ganada por el agente, simplemente le resta una cantidad fija de energía.

Por otro lado, las partículas son creadas en respuesta a diversas características de paquetes capturados de la red, son creadas, por elección arbitraria, en el extremo derecho del mundo y se mueven de manera discreta, un espacio a la vez, hacia la izquierda hasta que salen del mundo o su energía se agota, estas contienen dos datos importantes:

- Tipo
- Energía

Las partículas son gobernadas por otro conjunto de reglas:

1. $parametro(\lambda) \implies crearParticula(\lambda)$
2. $fueraDeRango(\rho) \implies eliminar(\rho)$
3. $sinEnergia(\rho) \implies eliminar(\rho)$
4. $posicionX(\rho) = posicionX(\rho) - 1$

Estas reglas quieren decir lo siguiente:

1. Si se observa uno de los parámetros λ esperados en el flujo de paquetes, entonces se inserta una partícula de ese tipo en el mundo, la partícula siempre sabe qué tipo de característica originó su creación, esto por medio de una variable que las

partículas contienen que define el tipo de partícula que se crea, las partículas son creadas en una posición Y aleatoria, pero siempre en el borde derecho del mundo.

2. Si una partícula ρ se sale de los límites definidos del mundo, esta es eliminada, por el diseño del modelo y la forma en que se mueven las partículas, esto siempre ocurre cuando cruzan el límite izquierdo del mundo, es decir, cuando su coordenada X es menor a cero.

3. Cuando una partícula ρ se queda sin energía, esta es eliminada.

4. En cada tick del mundo, las partículas se mueven un espacio a la izquierda de donde estaban, hasta que se salgan de los límites o su energía se agote por completo.

En relación a la primera regla de las partículas, la lista de características buscadas en el flujo de paquetes es la siguiente:

Flags generales
Bits reservados, DF y MF.
Protocolos
ICMP, TCP, EGP, UDP y OSPF.
Puertos
FTP (21), SSH (22), Telnet (23), SMTP (25), Time (37), DNS (53), HTTP (80), RTelnet (107), NNTP (119), NTP (123), IMAP (143), SNMP (161), BGP (179), IRC (194), SNMP (199) y HTTPS (443).
Flags TCP/UDP
CWR, ECN-Echo, URG, ACK, PSH, RST, SYN, FIN.

Estas características fueron elegidas arbitrariamente pero tratando de considerar aquellas que mejor podrían indicar la presencia de un ataque en progreso, por ejemplo, un exceso de flags SYN podría indicar que se está efectuando un ataque tipo Probe,

$$N_x = \Phi - i\epsilon$$

Figura 3.2: Ecuación para determinar el aporte energético de una partícula

que toma ventaja del *TCP Three-way Handshake* para averiguar qué puertos están abiertos.

Cuando una partícula ρ impacta a un agente x puede provocar un efecto positivo o negativo sobre él, dependiendo del resultado de la siguiente ecuación:

Donde N es el valor nutricional de la partícula para un cierto agente x, Φ es el máximo valor nutricional asociado a las partículas, i es el índice en el cuál ese tipo de partícula se encuentra en el código genético de x y ϵ es un parámetro de la partícula que representa su nivel actual de energía, o en otras palabras, representa la cantidad de energía que aún le queda a esa partícula.

Para el siguiente código genético de ejemplo:

$$\Theta = [16, 11, 17, 3, 22, ..., 8]$$

Si una partícula de tipo 17 es creada y colisiona con el agente que tiene este código genético el sistema debe determinar los parámetros que van a la ecuación, en este caso $\Phi = 90, i = 2, \epsilon = 7{,}15$, la elección de i es debido a que la partícula de tipo 17 se encuentra tercera en el código genético de este agente en particular y los índices en Netlogo comienzan en 0, por lo tanto la ecuación queda como sigue:

$$N_x = 90 - 2 \times 7{,}15$$

$$N_x = 75{,}7$$

Como esto corresponde a un número positivo, la partícula le otorga energía al agente, con esto se puede ver claramente que el comportamiento de un agente frente a una partícula depende principalmente del código genético del agente.

3.2. Sistema

En líneas generales, el sistema completo funciona de la siguiente manera, se crea una población de agentes de tamaño n, estos se inicializan en posiciones al azar dentro de un mundo de altura h y ancho w, los agentes tienen una energía inicial E_i, parámetro determinado de manera empírica, y un código genético $\Theta = [1, 2, 3, ..., 32]$, éste va desde 1 a 32 debido a que se escogieron 32 diferentes características a buscar en los paquetes, este código resulta de la permutación aleatoria de los números del 1 al 32, finalmente se guarda el estado completo de los agentes iniciales. El estado de los agentes que se guarda y se restaura incluye todo lo que a ellos concierne, es decir, su nivel de energía, posición en el mundo, su código genético y todas las demás variables que ellos contengan. Luego de estos preparativos, el sistema puede comenzar a funcionar. Netlogo ejecuta el código en *ticks*, es decir, como un solo gran *loop*, cada vuelta del algoritmo representa un *tick* y en este se realizan todas las acciones correspondientes a ese instante de la ejecución.

Cada *tick* del algoritmo realiza una serie de operaciones descritas a continuación:

1. Revisa si no ha terminado de leer el archivo con la captura de paquetes procesada, si aún queda archivo, entonces puede continuar.

2. Si no hay agentes con vida, utiliza el último estado de agentes que fueron guardados, por defecto estos son los agentes iniciales, pero cada vez que el sistema determina que ha comenzado un ataque, vuelve a guardar el estado actual de los agentes, el objetivo de esto es que si por alguna razón mueren todos los agentes, el sistema puede intentar recuperarse hasta que la población se estabilice, de otra manera, el sistema deja de operar correctamente y debe ser reiniciado, una segunda finalidad que tiene esto es que la población vuelva a su anterior estado inmediatamente después de terminado un ataque, esto busca evitar que, si hay varios ataques consecutivos separados por un corto tiempo, el sistema no sea capaz de reconocerlos debido a que la población aún se está recuperando.

3. Si aún queda parte de la captura de paquetes por leer, entonces el sistema lee una línea del archivo y genera los tipos de partículas que correspondan a las características halladas en el paquete.

4. El sistema actualiza la E_r, esta medida no es estática, inicialmente, la energía requerida para que un agente se reproduzca corresponde a un valor E_{r0} pero el sistema va aumentando esta energía gradualmente hasta llegar a un cierto valor E_{rf}. E_{r0} y E_{rf} son parámetros del sistema que fueron determinados empíricamente, el objetivo de esto, es que los agentes se adapten más fácilmente al ambiente en las etapas iniciales del sistema, esto se debe a que si los agentes comienzan requiriendo de una menor cantidad de energía para reproducirse, entonces su descendencia se adapta más rápidamente al ambiente, esto tiene como finalidad reducir el tiempo requerido para que el sistema alcance un equilibrio con el medio.

5. Cada agente realiza sus operaciones asignadas:

   a) Revisa si ha colisionado con una partícula para obtener energía.

   b) Realiza su gasto de energía por metabolismo.

   c) Si su energía es menor o igual a cero, el agente muere.

   d) Si puede reproducirse, entonces produce un hijo.

   e) Cambia el color del agente, el color de un agente depende de su nivel de energía, si es superior al 50 % de la E_r entonces el agente adquiere un color verde, si está entre 25 % y 50 % de la E_r entonces es amarillo y si es inferior al 25 % de la E_r el agente tiene un color rojo, esto tiene como fin simplemente ayudar a la visualización del modelo para la persona que lo está analizando, el sistema en sí no deriva ninguna información relevante del color de los agentes.

6. Las partículas realizan sus operaciones:

   a) Se mueven un espacio en la dirección que corresponda, en esta implementación es hacia la izquierda.

   b) Si su nivel de energía es igual o menor a cero, entonces la partícula es destruida.

Aclarando los puntos 1 y 3 anteriormente expuestos, se explica a continuación la forma en que son cargados los datos a Netlogo.

Para poder entregarle datos al sistema, se deben primero realizar un procesamiento de estos, ya que los datos no vienen en un formato que permita cargarlos directamente a Netlogo.

Primero se debe hacer una captura de paquetes de la red, para esto se utilizó el software Wireshark que permite guardar las capturas en texto plano, estas capturas contienen los bytes del paquete y su traducción a caracteres asociados.

Luego se debe hacer un procesamiento de estos datos, para ello se usan un par de herramientas hechas a medida para este trabajo, lo primero es eliminar la traducción a caracteres de los bytes del paquete, para esto se usa un script en *bash* que se aprovecha de las herramientas *sed* y de *awk* para dejar limpia la captura, sólo deja los bytes de cada paquete, estos paquetes se separan entre sí por un delimitador, agregándole otro delimitador que indica el fin de archivo, estos bytes listos se entregan a otra herramienta desarrollada en C++, esta se encarga de obtener los datos de cada paquete, es decir, esta herramienta es la que busca las características definidas anteriormente y con ellas construye un vector binario de largo 32, esto quiere decir, por ejemplo, que si el paquete es de tipo TCP y tiene la flag SYN marcada, entonces el vector correspondiente a él tendrá un 1 en la posición de TCP y un 1 en la posición de SYN, siendo 0 todas aquellas características no encontradas en el paquete, todo este proceso da como resultado un vector por paquete los que se escriben en un archivo en texto plano en el que cada vector ocupa una línea.

Finalmente este archivo se puede entregar directo a Netlogo, el sistema lee cada línea y por cada 1 que encuentra en un vector, crea una partícula con el tipo correspondiente al índice del valor encontrado.

3.3. Mejoras encontradas

Las mejoras buscadas apuntan a hacer más eficiente el clasificador, en ese sentido se realizaron los siguientes cambios al sistema:

- Cambios en los valores de los parámetros clave: Las pruebas realizadas determinaron que una variación en los parámetros del sistema da como resultado un

mejor desempeño del clasificador, los resultados de estas pruebas se encuentran en la sección 4.

- Modificación dinámica de la energía requerida para la reproducción: En un proceso que ya ha sido explicado con anterioridad, los agentes comienzan necesitando de una cierta energía E_{r0} para reproducirse, la cual va gradualmente aumentado hasta alcanzar una cierta energía final E_{rf}, de esta manera, al principio, los agentes se adaptan con mayor facilidad al medio.

- Adición de la función para guardar y restaurar el estado de los agentes: Esto permite que el sistema siempre se pueda recuperar, ya sea después de la eliminación de todos los agentes o al término de un ataque, restaurar la población a un estado anterior permite que el sistema siga funcionando.

- Uso de una nueva medida para el clasificador: Se agrega como medida para el clasificador la cantidad de partículas en el ambiente y el promedio de partículas durante una cierta cantidad de *ticks* como cota inferior.

3.4. Clasificador

Un paso extra que realiza el sistema en cada *tick* es el cálculo de las medidas estadísticas que permiten que funcione como un clasificador, esto es lo que se encarga de determinar si una situación es parte de un ataque o si es tráfico normal de la red, estas medidas son:

- Energía promedio de los agentes (E_{prom}).

- Población de los agentes (Pop_x).

- Cantidad de partículas en el ambiente (Pop_ρ).

Para realizar la clasificación, estas medidas se comparan con sus tendencias de una cierta cantidad de ticks, para E_{prom}, su línea de tendencia se calcula tomando los últimos 70 resultados del cálculo de la energía promedio, este resultado se multiplica luego por un factor de 0.95, con esto se consigue una línea equivalente al 95 %

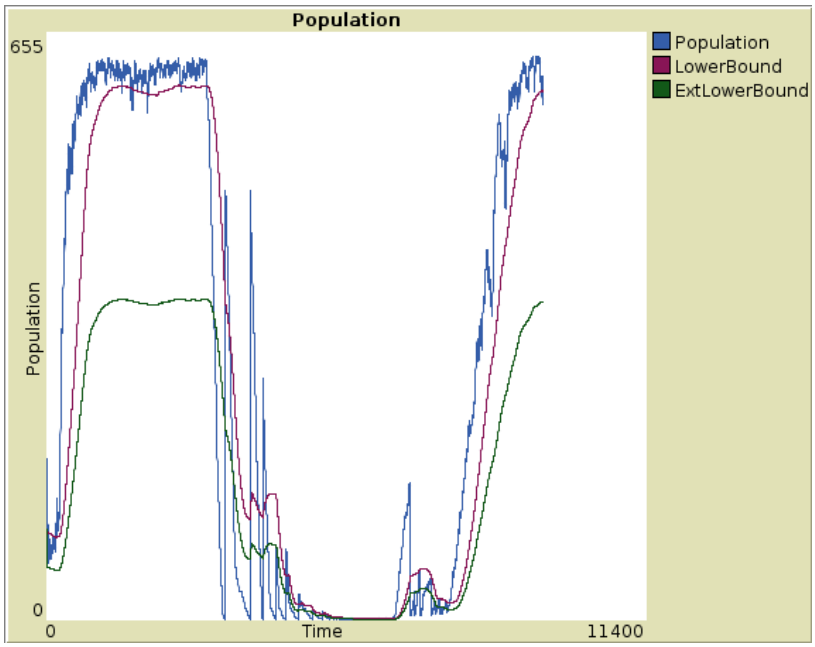

Figura 3.3: Gráfico de población con cota inferior y cota inferior extrema

de la tendencia de la energía promedio tomada con las últimas 70 muestras, esto corresponde a la cota inferior de la energía, fig: 3.4.

Para Pop_x, su tendencia corresponde a la población total de agentes de los últimos 100 ticks, este resultado se multiplica por 0.97 para obtener una línea de tendencia del 97 % de la población durante las últimas 100 muestras, esta es la cota inferior de la población. La población tiene también una segunda cota inferior, esta cota inferior extrema corresponde a la tendencia multiplicada por 0.6, el 60 % de la población de los últimos 100 ticks, el valor porcentual de esta cota inferior extrema con respecto a la tendencia fue elegido en base a la observación del comportamiento de la línea de tendencia de la población, en la figura fig: 3.3 se puede apreciar un cambio extenso, en tiempo, y notorio en la población promedio, pero en otros casos, se puede dar que el cambio sea súbito, al igual que su recuperación, esto ocasiona que un ataque que es demasiado corto pase desapercibido para esta medida en particular, la adición de esta cota inferior asegura que aunque el cambio sea de una duración relativamente corta (menor a 100 *ticks*), el sistema aún es capaz de detectarlo debido a que la población promedio cruza esta cota extrema.

La tendencia de Pop_ρ se calcula con la cantidad de partículas disponibles durante

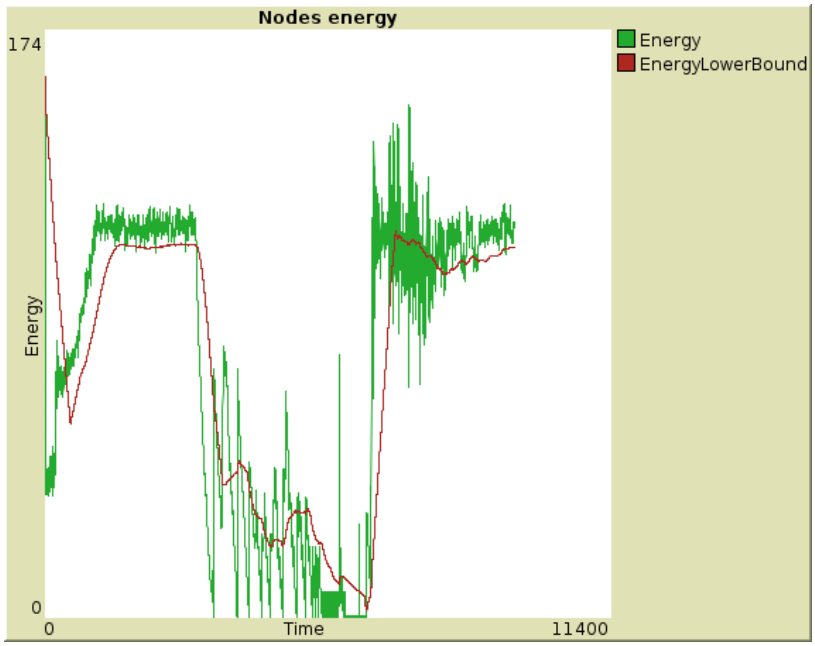

Figura 3.4: Gráfico de energía promedio con cota inferior

los últimos 100 ticks, este resultado se multiplica por un factor de 0.8 para obtener la cota inferior, pero también se multiplica aparte por 1.2, para obtener una cota superior, fig: 3.5, la razón de que la cantidad de partículas tenga una cota superior, a diferencia de las demaś medidas se debe a que la población y energía promedio de los agentes tienden a estabilizarse en un cierto máximo determinado por la cantidad de partículas que pueden habitar en el mundo al mismo tiempo, por lo tanto, una vez alcanzado este equilibrio, cualquier perturbación de esto sólo hará que los promedios de energía y población de agentes disminuyan, mientras que la cantidad de partículas en el ambiente está más limitada por las características buscadas en los paquetes, por lo tanto la cantidad de partículas puede aumentar si las características buscadas se encuentran más frecuentemente.

Los valores porcentuales de las cotas y umbrales fueron determinados tratando de mantenerse lo más cerca posible al valor real de la medida pero lo suficientemente alejados de esta tal que las cotas y umbrales no disparen alarmas con perturbaciones menores que pueden ser consideradas como algo normal en la red.

La razón de calcular estas cotas para cada medida es la forma en que el sistema realiza su clasificación, se define una ventana deslizante de un ancho específico para

cada medida, este ancho de ventana coincide con la cantidad de muestras usadas para determinar la cota de cada medida, es decir, 70 ticks para E_{prom}, 100 ticks para P_x y 100 ticks para P_ρ. Cada vez que alguna de las medidas sobrepasa su cota inferior, o en el caso de P_ρ también su cota superior, el sistema marca que hubo una desviación fuera de lo normal, si alguna de estas medidas sólo presenta desviaciones a lo largo de su ventana, sin ningún instante de normalidad, entonces el sistema clasifica la situación actual, y por ende la captura de paquetes, como una red que está siendo objeto de algún ataque o acto malicioso. La cota inferior extrema de la población tiene, por sí sola, la capacidad de declarar la ocurrencia de un ataque, si en cualquier momento la población actual cruza la cota inferior extrema, se declara inmediatamente una situación de ataque.

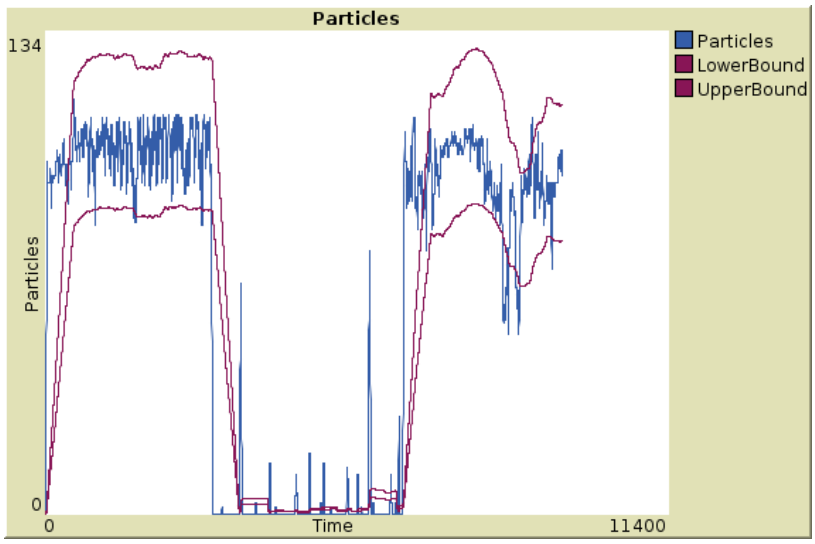

Figura 3.5: Gráfico con cantidad de partículas con cota superior e inferior

Todos estos valores fueron determinados en base a pruebas cuyos resultados se pueden ver un la tabla 3.6.

Para las pruebas se tomaron diferentes valores posibles para las cotas y se evaluaron sus resultados al clasificar las capturas de paquetes, de estos resultados se tomaron los 10 mejores y se realizó un conteo de la cantidad de veces que aparecía cada valor entre los que fueron elegidos, el valor final sería, en cada caso, aquél que hubiera aparecido más veces entre esos 10 mejores.

De igual manera se evaluaron las opciones para los tamaños de las ventanas deslizantes como se puede ver en la tabla 3.7.

Parámetro	**Valores obtenidos**	**Valor final**
Cota inferior para E_{prom}	0.85, 0.9, 0.95, 0.97 (1:3:5:1)	0.95
Cota inferior para Pop_x	0.9, 0.95, 0.97 (4:1:5)	0.97
Cota inferior extrema para Pop_x	0.4, 0.5, 0.6 (3:3:4)	0.6
Cota inferior para Pop_ρ	0.8, 0.9 (7:3)	0.8
Cota superior para Pop_ρ	1.1, 1.2, 1.3 (2:6:2)	1.2

Figura 3.6: Tabla de valores para las cotas

Parámetro	**Valores obtenidos**	**Valor final**
Ventana para E_{prom}	50, 70, 80 (2:5:3)	70
Ventana para Pop_x	80, 90, 100, 120 (2:3:4:1)	100
Ventana para Pop_ρ	100, 110 (6:4)	100

Figura 3.7: Tabla de valores para las ventanas deslizantes

Capítulo 4

Resultados

El objetivo de la experimentación es obtener el conjunto de parámetros que permitan que el clasificador alcance el mejor desempeño posible, para esto y como primer paso, se determinaron los parámetros del sistema que tienen mayor influencia en el funcionamiento del clasificador, estos parámetros son:

- La energía inicial necesaria para que un agente se reproduzca (E_{r0}).
- La cantidad de agentes iniciales (x_i).
- La energía de los agentes al nacer (E_b), corresponde a la energía inicial de un agente creado como resultado de la reproducción de otro agente.
- La energía gastada por los agentes durante el proceso de reproducción (E_{proc}).
- La energía final necesaria para que un agente se reproduzca (E_{rf}).
- La energía inicial de los agentes creados (E_i), corresponde a la energía con la que son creados los agentes de la primera generación del sistema, aquellos cuyo código genético es completamente aleatorio.
- La energía gastada para mantener activo el metabolismo de los agentes (m).
- El máximo valor nutricional que puede aportar una partícula a un agente (Φ).
- La pérdida lineal de valor nutritivo de los tipos de partículas respecto a los agentes (ϵ).
- La dimensión X del mundo que habitan los agentes (*max-pxcor*).
- La dimensión Y del mundo que habitan los agentes (*max-pycor*).

Para probar los parámetros se tomaron los del modelo de base y, utilizando un cierto rango individual, se iteró sobre todas las posibles combinaciones obtenidas. Para realizar la prueba exhaustiva de parámetros, se utilizó una herramienta llamada *BehaviorSpace*, ésta viene contenida como parte del software NetLogo, el *BehaviorSpace* se encarga de realizar todas las pruebas necesarias y reporta las variables que se le indican al terminar. En este sentido, podemos decir que el *BehaviorSpace* sólo simplifica el trabajo de determinar los parámetros que mejores resultados entregan, sin embargo no realiza ninguna optimización del modelo por cuenta propia, eso se debe hacer con un análisis posterior de los datos obtenidos.

Para determinar la mejor combinación de parámetros, se obtuvieron las 10 combinaciones que mejores resultados reportaron y se realizó un promedio ponderado de sus parámetros, la tabla 4.1 resume los resultados obtenidos.

Parámetro	**Valor de base**	**Valores obtenidos**	**Valor final**
E_{rf}	250	200, 300 (9:1)	210
x_i	300	75, 100, 125 (2:6:2)	100
E_b	50	100, 125 (1:9)	122.5
E_{proc}	100	25, 50 (1:9)	47.5
m	0.6	0.5, 0.75, 1 (4:5:1)	0.675
ϵ	5.5	5.5, 7, 9.5 (4:3:3)	7.15
E_{r0}	No aplica (250)	100 (1)	100
E_i	150	100, 150, 200 (1:6:3)	160
Φ	90	75, 100, 125 (5:4:1)	90
max-pxcor	32	24 (1)	24
max-pycor	16	24 (1)	24

Figura 4.1: Tabla de comparación de parámetros entre el sistema de base y el nuevo

La tabla contiene, como primera columna, el parámetro que se está probando, como segunda columna el valor original del parámetro en el modelo de base, como tercera columna los nuevos valores obtenidos en el modelo actualizado junto con sus ponderaciones y como cuarta columna el valor final ponderado de los nuevos valores obtenidos.

Estos parámetros se obtuvieron con pruebas de pequeña a mediana escala, en términos de cantidad de paquetes por captura y cantidad total de capturas. Las pruebas consistieron en tomar capturas de paquetes de la red normal y bajo ataques

y de verificar si el sistema era capaz de detectar correctamente eventos de ataque y de normalidad.

El parámetro E_{r0} no aplica para el modelo de base, ya que esta fue una modificación que es parte de las mejoras del nuevo modelo, con estos parámetros se realizaron pruebas más completas para determinar si efectivamente se desempeñan mejor a mayor escala, con los resultados de estas pruebas se calcularon las 4 medidas principales que determinan si un clasificador es mejor que otro:

- Precisión: Cuantas capturas clasificadas como ataque efectivamente lo son.

$$\frac{VP}{VP + FP}$$

- Sensitividad: Cuantos ataques, del total de ataques, fueron clasificados como tales.

$$\frac{VP}{VP + FN}$$

- Exactitud: Cuantas clasificaciones correctas fueron hechas del total de capturas.

$$\frac{VP + VN}{VP + FP + VN + FN}$$

- Especificidad: Cuantos no-ataques, del total de no-ataques, fueron clasificados correctamente.

$$\frac{VN}{VN + FP}$$

Para calcular estas 4 medidas, se deben primero obtener los verdaderos positivos, falsos positivos, verdaderos negativos y falsos negativos, todas las pruebas para esto se realizaron con capturas de paquetes obtenidas de la red de la Universidad de Concepción.

Para las capturas que no contienen ataque se realizaron algunas capturas individuales en días aleatorios y además se hizo una captura mucho más grande que las demás, la que después fue particionada para convertirla en más capturas de un tamaño manejable. Se realizaron un total de 84 diferentes capturas entre Abril y Octubre

del 2017, estas se clasifican como se muestra en la tabla 4.2. Todas las capturas fueron realizadas en la red Wi-Fi *SalaEstudioInformatica2* del Edificio de Sistemas de la universidad, red elegida por ser ampliamente utilizada por los alumnos.

Capturas normales	60
Capturas con ataque DoS	10
Capturas con ataque Probe	10
Capturas con ataque MITM	4

Figura 4.2: Tipos de capturas realizadas

La elección de probar con ataques tipo DoS se debe a una reciente alza de estos, incluyendo nuevas opciones de ataque mediante el uso de dispositivos que hacen uso de la IoT (*Internet of Things*), [20]. Los ataques Probe fueron elegidos ya que estos sirven para recopilar información acerca del objetivo, por lo tanto son usualmente una de las primeras herramientas usadas para descubrir posibles superficies de ataque a un sistema. Por último, la elección de probar con ataques tipo MITM es debido a que este tipo de ataque sigue siendo usado con regularidad para interceptar tráfico de una red [7].

Para realizar estos ataques se procedía de distintas formas. Para el ataque de denegación de servicio se elige una dirección IP aleatoria que puede no existir para evitar causar algún daño, el ataque fue lanzado con la herramienta *hping3* [11] que viene incluida con Kali Linux, esto realiza ataques de denegación de servicio inundando al servidor con peticiones, tantas como sea posible.

Para el ataque Probe se utilizó la herramienta *nmap* [12], que viene como parte de Kali Linux, *nmap* permite obtener una enorme cantidad de información de la máquina objetivo, en particular, los puertos abiertos, qué servicios están siendo ejecutados en ellos y sus versiones, puede incluso determinar la versión de sistema operativo que utiliza la máquina y además puede ofuscar sus peticiones para tratar de saltarse algunos sistemas de seguridad que pudieran estar presentes.

Para el ataque de *Man-In-The-Middle* se utilizó una herramienta llamada *bettercap* [1], esta es una herramienta desarrollada con el fin de reemplazar a ettercap que era la herramienta comúnmente usada para estos fines.

Bettercap otorga una serie de utilidades interesantes como la posibilidad de elegir

el método para lograr el *MITM*, además de la posibilidad de realizar algún tipo de ssl-stripping [15, 8], esto permite que la comunicación dirigida a través de *HTTPS (Hypertext Transfer Protocol Secure)* que usualmente es segura debido a su encriptación, pase a ser simplemente *HTTP (Hypertext Transfer Protocol)*, logrando de esta manera ver su contenido.

A continuación se presentan los resultados obtenidos por ambos modelos, los valores se presentan en términos de evento detectados:

Verdaderos positivos	120
Falsos positivos	70
Verdaderos negativos	230
Falsos negativos	0

Figura 4.3: Modelo de base, positivos y negativos, por cantidad de eventos

Verdaderos positivos	120
Falsos positivos	50
Verdaderos negativos	250
Falsos negativos	0

Figura 4.4: Modelo nuevo, positivos y negativos, por cantidad de eventos

Estos resultados determinan que las medidas sean las siguientes:

Precisión	63.15 %
Sensitividad	100 %
Exactitud	83.33 %
Especificidad	76.66 %

Figura 4.5: Modelo de base, medidas estadísticas

Precisión	70.58 %
Sensitividad	100 %
Exactitud	88.09 %
Especificidad	83.33 %

Figura 4.6: Modelo nuevo, medidas estadísticas

De estos resultados se puede calcular que el nuevo modelo desarrollado logró una mejora promedio de un 4.715 %, con la máxima diferencia de un 7.43 % en la precisión y una mínima de 0 % en la sensitividad.

De estos resultados podemos concluir que, con respecto a las condiciones medidas en las pruebas realizadas, el modelo nuevo alcanzó un cierto nivel de mejora al compararlo con el modelo de base, es decir, el modelo nuevo fue capaz de discernir mejor si una situación correspondía o no a un ataque. Para poder determinar si el modelo nuevo es superior al de base sería necesario realizar una serie de pruebas más completas, con conjuntos de datos más grandes.

Las pruebas también ayudaron a determinar otro parámetro del sistema que no ha sido listado en las tablas porque en un principio no se le consideró importante, al comenzar la ejecución del sistema los agentes iniciales necesitan de un período de adaptación, esto representa lo que tarda la especie en adaptarse a lo que supone que son las condiciones normales del medio, en el modelo nuevo, bastan entre 1000 y 1500 *ticks* para que los agentes se encuentren en condiciones de reaccionar a cambios en el ambiente, esto contrastado con el modelo inicial que podía necesitar hasta 3000 *ticks* para adaptarse al ambiente inicial. Esto significa que en el modelo antiguo las detecciones realizadas antes de los 3000 *ticks* tienen una mayor probabilidad de ser falsos positivos debido a que los agentes aún se están adaptando, en cambio en el modelo nuevo, este número está más cercano a los 1500 *ticks*.

Capítulo 5

Conclusiones

Para concluir y resumiendo brevemente lo realizado, partiendo de la base de un sistema desarrollado con anterioridad, se investigaron, implementaron, probaron y midieron mejoras al respecto, estas mejoras corresponden a la calidad del sistema clasificador, se buscaba que la capacidad que tiene el sistema para discernir entre diferentes situaciones mejorara (ataque o no-ataque) y se obtuvo una mejora de alrededor del 5 % en las medidas relevantes, estas mejoras deben ser situadas en el contexto de las pruebas que se hicieron, para poder obtener una conculsión definitiva sería necesario realizar pruebas más exhaustivas al respecto.

La mayoría de los problemas que surgieron durante esta investigación tienen que ver con el uso del software Netlogo, este utiliza su propio lenguaje que opera de manera un tanto distinta a otros conocidos, de igual manera la sintaxis no es particularmente simple, la gran ventaja es que, pese a las falencias mencionadas, sigue siendo un lenguaje rápido para desarrollar modelos y permite desplegar elementos visuales con una facilidad que no existe en lenguajes de más bajo nivel.

El procesamiento inicial de las capturas de paquetes también supuso un desafío, estos paquetes vienen con una buena cantidad de datos que no son necesarios para el análisis y por ello hubo que partir desarrollando un script en bash que utiliza una combinación de sed y awk para limpiar los paquetes, para posteriormente entregarlos a un programa desarrollado en C++, el cual se encarga de transformar esos paquetes en vectores binarios.

En cuanto al trabajo futuro, siempre se pueden seguir realizando mejoras al modelo, entre las cosas que quedaron sin explorar está la opción de utilizar la variabilidad genética de la población para obtener alguna medida significativa para el clasificador, cambiar la forma en que las partículas alimentan a los agentes, por ejemplo se puede hacer que las partículas sean eliminadas al ser consumidas por un agente, o podría

cambiarse el patrón de movimiento de estas, agregar comportamientos más complejos a los agentes, quizá si estos se pudieran mover se apreciarían interesantes efectos demográficos. También es importante hacer pruebas más extensas con diversos tipos de ataques y en otras redes.

Lo más importante en cuanto a lo que queda por hacer, es una implementación real del modelo, Netlogo es simplemente para construir, probar y afinar el modelo, para poder hacer una verdadera implementación de este se debe cambiar a un lenguaje que lo permita, se postula como idea a futuro hacer una implementación en C++ debido a que es rápido en ejecución, o desarrollarlo en Python que tiene la ventaja de poseer librerías para casi todo lo que se desee hacer y es además relativamente rápido de implementar en este lenguaje con el coste de un poco más de demora en tiempo de ejecución si se compara con C++.

Bibliografía

[1] Bettercap. Bettercap stable documentation. https://www.bettercap.org/, 2017.

[2] Christian Blum, José A. Lozano, and Pedro Pinacho Davidson. *An artificial bioindicator system for network intrusion detection.* Number 21. Artificial Life, 2012.

[3] Catania C. and Garcia C. *Automatic network intrusion detection: Current techniques and open issues.* Number 38. Computers & Electrical Engineering, 2012.

[4] CISCO. The zettabyte era: Trends and analysis. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html, 2017.

[5] Stopel D., Moskovitch R., Boger Z., Shahar Y., and Elovici Y. *Using artificial neural networks to detect unknown computer worms.* Number 18. Neural Computing and Applications, 2009.

[6] Exploit DB. Exploit database statistics. https://www.exploit-db.com/exploit-database-statistics/, 2017.

[7] Erik de Jong and Frank. Lessons learned from a man-in-the-middle attack. https://www.fox-it.com/en/insights/blogs/blog/fox-hit-cyber-attack/, 2017.

[8] Leonardo Nve Egea. Offensive: Exploiting dns servers changes blackhat asia 2014. https://www.slideshare.net/Fatuo__/offensive-exploiting-dns-servers-changes-blackhat-asia-2014, 2014.

[9] Daniel Farr. *Indicator Species. in Encyclopedia of Environmetrics (eds. A H El-Sharaawi and W W Piegorsch).* John Wiley & Sons, Ltd., 2002.

[10] Stolfo SJ. Lee W. *Data mining approaches for intrusion detection*, volume 7. USENIX Association, 1998.

[11] Kali Linux. hping3 | penetration testing tools. https://tools.kali.org/information-gathering/hping3, 2017.

[12] Kali Linux. Nmap | penetration testing tools. https://tools.kali.org/information-gathering/nmap, 2017.

[13] W. Lu and I. Traore. *Detecting new forms of network intrusion using genetic programming.* Number 20. Computational Intelligence, 2004.

[14] Roesch M. *SNORT – lightweight intrusion detection for networks.* USENIX Association, 1999.

[15] Moxie Marlinspike. Moxie marlinspike software sslstrip. `https://moxie.org/software/sslstrip/`, 2017.

[16] Mahoney MV and Chan PK. *Learning nonstationary models of normal network traffic for detecting novel attacks.* Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, 2002.

[17] Mahoney MV and Chan PK. *PHAD: packet header anomaly detection for identifying hostile network traffic.* Florida Institute of Technology, 2004.

[18] Pedro Pinacho, Iván Pau, Max Chacón, and Sergio Sánchez. *An ecological approach to anomaly detection: the EIA model.* Number 7597. ICARIS LNCS, 2015.

[19] Staniford S, Hoagland JA, and McAlerney JM. *Practical automated detection of stealthy portscans.* J Comput Secur, 2002.

[20] Symantec. 2017 internet security threat report. `https://www.symantec.com/content/dam/symantec/docs/reports/gistr22-government-report.pdf`, 2017.

[21] Lindqvist U and Porras P. *Detecting computer and network misuse through the production-based expert system toolset (P-BEST).* Proceedings of the 1999 IEEE symposium on security and privacy, 1999.