Utilizando Inteligencia Artificial para la detección de Escaneos de Puertos

Amador, Siler., Arboleda, Andrés y Bedón, Charles.
{samador,aarboleda,cbedon}@unicauca.edu.co
Universidad del Cauca

Resumen— La amplia gama de ataques a redes de computadores y sus variantes han hecho de la detección de intrusos una ardua tarea, sobre todo, teniendo en cuenta que además de la implantación de herramientas diseñadas para tal cometido, es necesario actualizar constantemente la base de conocimientos o reajustar los parámetros de configuración de dichas herramientas si la red o el contexto de operaciones cambia en determinado momento. Es por esto que la incorporación de técnicas de inteligencia artificial para adicionar cierto grado de adaptabilidad a los detectores de intrusos está tomando cada vez más fuerza. En el siguiente artículo se describirá el análisis realizado y los resultados obtenidos dentro del proyecto "Sistema de Detección de Intrusos Utilizando Inteligencia Artificial" para introducir un elemento inteligente, basado en redes neuronales, orientado a la detección del problema específico del escaneo de puertos, usando como base el IDS (*Intrusion Detection System* – Sistema de Detección de Intrusiones) de código abierto Snort™.

Índice de Términos— Detección de Intrusos, Escaneo de Puertos, IDS, Inteligencia Artificial, Redes Neuronales, Snort.

I. INTRODUCCIÓN

Debido a la masificación del uso de Internet, la publicación de información que permite el acceso ilegal a datos compartidos en redes privadas, es muy simple, lo cual ha incrementado el número de incidentes de seguridad en los últimos años en América latina, a pesar de la aparente leve disminución a nivel mundial [1]. Por supuesto, el interés en tratar de detener o protegerse de estas intrusiones ha aumentado igualmente, y es por esto que nuevas iniciativas están siendo generadas. En la propuesta "Sistema de Detección de Intrusos Utilizando Inteligencia Artificial" [2], se pretende detectar un ataque desde su propia gestación: El reconocimiento de vulnerabilidades. La idea es trabajar sobre una infraestructura de recolección de información y detección, probada y optimizada, para reutilizar ciertos elementos y mejorar los ya existentes en el campo de la detección de escaneos de puertos; El NIDS (Network IDS) Snort™ es la herramienta ideal, teniendo en cuenta que es el detector de intrusos de código abierto de uso más difundido, con una comunidad activa y reconocida efectividad. Sin embargo usa un tipo de detección basado en uso indebido, es decir, detecta intrusiones analizando *firmas de ataque*, que son fijas, de tal suerte que una variante de ataque no registrada podría confundirlo evitando que sea reportada o provocar la generación de falsas alarmas.

Este proyecto no muestra una solución radicalmente innovadora, dentro del estado del arte de los Sistemas Detectores de Intrusos (IDS) que utilizan tecnologías de Inteligencia Artificial (IA). Pero es el primer paso a nivel local en una línea de investigación que tiene futuro, de la mano de la filosofía del Software Libre.

II. ESCANEO DE PUERTOS

Un escaneo de puertos (*portscan*) es una técnica de exploración que pretende hallar qué servicios están siendo ofrecidos por una red o servidor, consiste en realizar conexiones o intentos de conexión a diferentes puertos (TCP o UDP) en la víctima esperando obtener respuesta de alguno o algunos de ellos e inferir qué aplicación o servicio está escuchando en dicho puerto. Así por ejemplo, si recibe una respuesta del puerto 22 supondrá que se trata del servicio de SSH, aunque probablemente sea un servidor Web el que esté escuchando peticiones en aquel puerto si el administrador del host así lo ha configurado.

A pesar de que es sabido que esta actividad es comúnmente el preludio para un ataque de mayores proporciones, no es penalizada por la ley de Colombia ni de Estados Unidos ya que se asocia con el caso análogo en el que un ladrón golpea la puerta de una casa para ver si está abierta o no, pero permanece afuera de ella; sin embargo, en el ambiente de la seguridad informática es considerada como un ataque.

Existe una gran cantidad de tipos de escaneo [3] que pueden basarse en los protocolos TCP o UDP, diferenciándose básicamente por las banderas de protocolo utilizadas (como SYN, ACK o RST aplicables sólo a TCP), pero los dos tipos de escaneo mas comunes son el TCP Connect y el TCP SYN los cuales se muestran en la Fig. 1. El TCP Connect utiliza el proceso de conexión convencional del protocolo TCP conocido como triple *handshaking* o saludo triple, llamado así por los tres mensajes que se intercambian al inicio de una nueva conexión (SYN, SYN_ACK y ACK). El escaneo TCP SYN o semiabierto no establece una conexión por cada puerto, es mas rápido y difícil de detectar.

Todos los tipos de escaneo tienen como denominador común dos elementos: la cantidad y la temporalidad. El primero se refiere al número de intentos de conexión que se hagan a un host (tantos intentos como puertos se desee escanear, puede estar entre 1 y 65535) y el segundo a cuanto tiempo promedio separa un intento de otro (muy corto, milisegundos, si se desea cubrir un buen rango de puertos). Estos elementos permanecen aún si el escaneo es distribuido, en tiempo o en espacio. En tiempo significa que se escanea cierto rango de puertos en un momento y después de un lapso prudencial, se escanea el resto. En espacio quiere decir que no se hace desde un mismo host, sino que se eligen varios para lanzar el ataque. De hecho, en el escaneo de tipo *Decoy* [3] o escaneo con señuelos se simula una exploración desde diferentes máquinas para confundir al IDS, pero realmente se hace desde uno solo.

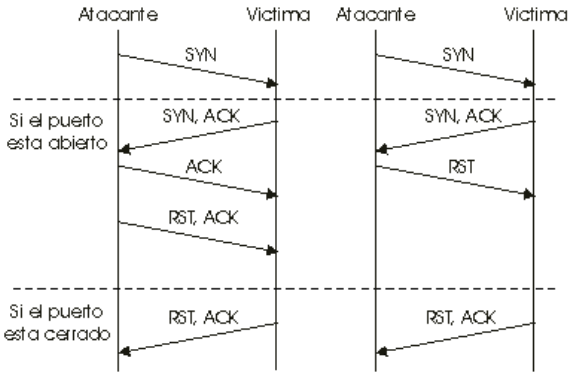

Fig. 1
ESCANEO TCP CONNECT (IZQUIERDA) Y TCP SYN (DERECHA)

Estas características son la base del análisis que se efectuó, partiendo además de la premisa que al administrador no le interesa saber si el escaneo es de tipo XMAS tree, TCP SYN, TCP Connect o cualquier otro, sino que está siendo escaneado y que la fuente ha sido identificada con un grado aceptable de precisión.

III. REDES NEURONALES

El funcionamiento exacto del cerebro humano sigue siendo un misterio. En detalle, el elemento más básico del cerebro humano es un tipo específico de célula, llamado neurona, que a diferencia del las células del resto del cuerpo no se regenera. Se asume que estas células son las que nos proveen las capacidades de recordar, pensar, y aplicar experiencias anteriores a cada una de nuestras acciones.

Estas Redes Neuronales Artificiales (RNA) sólo intentan replicar los elementos más básicos de este complicado, versátil, y poderoso organismo. Lo hacen de una manera primitiva. Pero para el programador que está intentando solucionar un problema determinado, la computación neuronal nunca fue sobre replicar cerebros humanos. Es sobre máquinas y sobre una nueva manera de solucionar problemas [16].

Entre las diferentes tecnologías de Inteligencia Artificial (lógica difusa, técnicas de búsqueda como algoritmos genéticos, etc) se escogió las Redes Neuronales Artificiales porque era la más madura en el campo de la detección de intrusos, y porque

las herramientas de código abierto para el entrenamiento y generación de código estaban a disposición. Un estudio más minucioso hubiera requerido implementar todas las tecnologías existentes y por medio de pruebas comparativas concluir cuál era la mejor, sin embargo, el alcance del proyecto no fue definido para tal efecto.

Al trabajar con Redes Neuronales, no se utiliza la programación tradicional, en la cual se especifican un conjunto de instrucciones que se ejecutan de forma secuencial. En cambio implica la creación de redes paralelas masivas y el entrenamiento de esas redes para solucionar problemas específicos.

A. La neurona artificial

La unidad básica de las Redes Neuronales Artificiales, la neurona artificial, simula las funciones básicas de las neuronas biológicas. La operación de una neurona artificial o unidad se puede resumir de la siguiente manera: las señales son presentadas a la entrada, cada señal es multiplicada por un número, o peso, que indica su influencia en la salida de la neurona; y se realiza la suma ponderada de las señales que produce un nivel de actividad. Finalmente, a dicho nivel de actividad se le aplica la *función de activación,* y la unidad produce una determinada respuesta de salida.

B. Función de activación

Las funciones de activación también llamadas funciones de transferencia, son las responsables de determinar la forma y la intensidad de variación de los valores transmitidos de una neurona a otra. Entre las funciones de activación mas utilizadas esta la **función sigmóide (*sigmoid* o *logistic*)**, ésta permite una transición gradual y no lineal entre dos estados. Su gráfica se muestra en la Fig. 2.

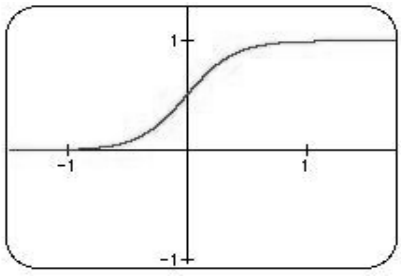

Fig. 2
FUNCIÓN DE ACTIVACIÓN SIGMÓIDE

C. Topología de una Red Neuronal

Basándose en el patrón de conexiones (arquitectura), las RNA pueden ser agrupadas en dos categorías: redes no recurrentes y redes recurrentes.

En las redes no recurrentes (*feed-forward*), las señales de entrada son simplemente transformadas en señales de salida, no hay conexiones cerradas (*loops*) y se sigue siempre un flujo continuo hacia el frente. En cambio en las redes recurrentes (*recurrent* o *feedback*), las señales son alteradas en diversas transiciones de estado, siendo la salida alimentada también de la entrada, igualmente se tienen conexiones cerradas debido a las realimentaciones.

D. Aprendizaje

El aprendizaje se puede clasificar en supervisado y no supervisado. Las redes con aprendizaje supervisado requieren de una serie de datos de ejemplo, llamados *sets* (conjuntos) de entrenamiento, que le muestran a la red qué valores de salida se deben obtener para ciertos valores de entrada, así la red puede generalizar luego de ser entrenada y clasificar un valor de entrada aunque este no se encuentre entre los datos de ejemplo. Las redes con aprendizaje no supervisado carecen de ejemplos en los cuales basarse, su ejecución es más lenta y se aplican principalmente en la identificación de características de conjuntos de datos, como por ejemplo hallar cual es el valor que más se repite.

E. Topologías utilizadas

En la implementación de soluciones con redes neuronales, son pocos los factores de diseño que se pueden determinar a partir del análisis del problema. Para el caso de las redes neuronales, sólo se puede determinar con seguridad si la red a utilizar será de entrenamiento supervisado o no supervisado, para el resto de factores se hace necesario el uso de métodos de ensayo y error. Durante el proyecto fue necesario probar diferentes topologías de red con sus propios algoritmos de entrenamiento, así como diferentes conjuntos y

codificaciones para las entradas de la red con el fin de obtener una convergencia aceptable en el error.

En el prototipo definitivo del proyecto se utilizaron dos topologías de redes con entrenamiento supervisado: la red MLP (Multi-Layer Perceptron) y la red Elman.

La red MLP básicamente consiste de un conjunto de unidades sensoriales que constituyen la capa de entrada, una o más capas intermedias u ocultas y una capa de salida de nodos computacionales. La señal de entrada se propaga de una manera no recurrente.

La red Elman [15] por su parte, es una red parcialmente recurrente, posee una arquitectura que maneja el contexto de los patrones de entrada. De modo que un patrón depende del anterior y así sucesivamente. Esto significa dar al sistema procesador propiedades dinámicas que responden a secuencias temporales, siendo esto optimo para entradas que son naturalmente presentadas en secuencia. La red Elman se compone de las mismas capas que una red MLP, pero adicionalmente posee una capa de contexto asociada a las capas ocultas.

IV. PREPROCESADORES EN SNORT™

Como se señala en los documentos "Snort™ Diagrams for Developers" [4] y "El IDS Snort™ ante un ataque de TCP reset" [5], el NIDS Snort™ se compone básicamente de cuatro módulos a saber (Ver Fig. 3):

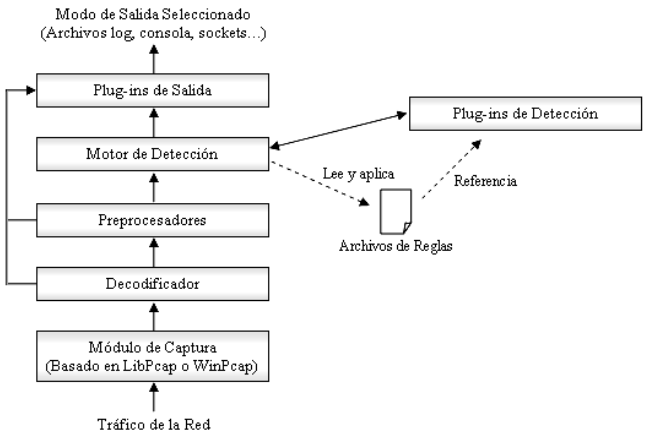

Fig. 3
DIAGRAMA EXPLICATIVO DE LOS MÓDULOS DEL SNORT

A. *Captura de Paquetes*

Usando la librería WinPcap o LibPcap, captura los paquetes y analiza diversos tipos de protocolos desde capa de enlace hasta transporte y los organiza en estructuras para su posterior análisis

B. *Preprocesadores*

Son plugins encargados de detectar por medio de un tratamiento algorítmico un comportamiento o extraer información de cada paquete que llega.

C. *Motor de Detección y evaluación de Reglas*

Se podría definir como el corazón de Snort, no se puede quitar, y relaciona las características de un paquete con las reglas viendo si hay alguna coincidencia para alertar.

D. *Plugins de Salida*

Tiene como objeto formatear las salidas que genere Snort, esto permite guardar las alertas y la información relacionada ya sea en bases de datos, en archivos de texto plano o archivos en formatos especiales.

En el proyecto se decidió implementar un preprocesador para la detección, frente a la inconveniencia técnica de generar reglas de una forma dinámica que permitan identificar los ataques comunes así como sus pequeñas variantes. Además estos componentes permiten manejar una carga lógica más compleja que la de simples sentencias organizadas con cierto orden sintáctico.

Es de resaltar que gracias a la gran modularidad y escalabilidad de Snort, fue posible integrar con éxito el componente de inteligencia artificial.

V. ESTADO DEL ARTE

Esta es una visión general del estado del arte de trabajos con redes neuronales artificiales para la detección de intrusos de acuerdo a lo descrito en [6].

El primer modelo de detección de intrusos basado en redes neuronales lo realizó K. Fox et al [7] en 1990 como método para crear perfiles de comportamiento de usuarios. Al igual que H. Debar

y B. Dorizzi en [8] (1992), utilizan redes neuronales para predecir el siguiente comando basado en una secuencia de comandos previos ejecutados por un usuario. El aprendizaje lo realizan mediante redes neuronales recurrentes (parte de la salida se realimenta como entrada a la red en la siguiente iteración) por lo que la red está continuamente observando y tiene la capacidad de "olvidar" comportamientos antiguos. Debar y Dorizzi presentan un sistema de filtrado basado en redes neuronales recurrentes que actúa para filtrar los datos que no corresponden a la tendencia observada en el comportamiento de las actividades de usuarios.

Más tarde (1997), los brasileños Cansian et al., desarrollan el sistema SADI (Sistema Adaptativo de Detecção de Intrusão), que usa la red neuronal para identificar el comportamiento intrusivo de patrones capturados de una red [9][10].

En los años subsiguientes surgen variadas iniciativas en este campo, estando entre ellas la propuesta de Bivens y sus compañeros en el 2002. Esta se basa en el método de tiempo de ventana (time-window). Este método les permite reconocer ataques multi-paquete más largos [11]. También utilizan un modelo híbrido de perceptrón multicapa y mapas autoorganizativos, para la detección de anomalías. En el 2003, en la Universidad de Ohio se desarrolló un IDS de red llamado INBOUNDS (Integrated Network-Based Ohio University Network Detective Service) donde un módulo de detección de anomalías basado en análisis estadístico se sustituyó por otro que utiliza mapas autoorganizativos [12].

Otras propuestas han sido planteadas en los últimos tres años, como la del brasileño Fábio Bombonato en la que desarrolla un prototipo llamado BEHOLDER, el cual, mediante una red neuronal MLP, basándose en el motor de captura de SADI, detecta escaneos de puertos. Sin embargo, la solución que propone no presenta resultados convincentes y las entradas seleccionadas para la red neuronal no aprovechan la capacidad de abstracción de la misma [13].

En Colombia, también han surgido propuestas en este campo. En el 2003, Efraín Torres Mejia [14] desarrolla un IDS para la detección de ataques en el protocolo HTTP. Utilizando una red Elman, analiza las cadenas de caracteres que componen los mensajes HTTP en busca de patrones de ataque conocidos. Este trabajo fue presentado en las III Jornadas Nacionales de Seguridad Informática ACIS 2003.

VI. PortscanAI

El preprocesador PortscanAI es el componente de desarrollo del proyecto, integra una red neuronal que le permite detectar escaneos de puertos con una considerable reducción en la cantidad de falsos positivos (falsas alarmas) y falsos negativos (ataques sin detectar), adicionalmente puede arrojar estadísticas y gráficas, como característica opcional, para el análisis del tráfico de red por parte del administrador.

La primera consideración a tomar para la implementación de este preprocesador es que debe hacer un análisis *stateful*, es decir que tiene que tener en cuenta tanto el paquete actual como los anteriores. En segundo lugar, se debe tener en cuenta que no todos los paquetes necesitan ser analizados, sólo aquellos que pretendan iniciar una conexión, que para el caso de TCP significaría aquellos con la bandera SYN. Dentro del grupo anteriormente mencionado se debe pesar los elementos reseñados en la sección de Escaneos de Puertos, la cantidad (cuántos son enviados) y la temporalidad (cada cuanto se efectúa esta operación). La tercera consideración es precisamente que debido a que no se puede analizar un historial de paquetes demasiado largo, por cuestiones prácticas y de rendimiento, se debe analizar por lo menos los que estén dentro de una ventana de tiempo, cantidad de intentos de conexión o algún parámetro fácilmente identificable.

Para cubrir las anteriores consideraciones se realizó un análisis de tráfico de red bajo condiciones de escaneo, generado con el software Nmap, y comparándolo con tráfico libre de este tipo de ataques. Gracias a este análisis se pudo identificar qué medidas son las más importantes a la hora de

detectar un escaneo de puertos. Esas medidas son: número de intentos de conexión enviados y recibidos por cada dirección IP, tiempo entre dichos intentos y cantidad de respuestas enviadas por una cierta dirección IP indicando que un puerto está cerrado.

Con estas medidas se pudieron generar distintos *sets* de entrenamiento para la red neuronal, ensayando diferentes combinaciones como entradas y observando el error producido por estas durante el entrenamiento. Inicialmente se probaron diseños de redes cuyas entradas no eran normalizadas y con una sola salida, sin lograr obtener una reducción adecuada del error (su error permanecía constante o aumentaba con los ciclos de entrenamiento) a pesar de los múltiples intentos por mejorar su desempeño. Luego de esto se optó por normalizar las entradas y colocar una salida binaria por cada uno de los estados que arrojara la red, siendo esta la opción que mejor se comportó. Las redes escogidas fueron: un Perceptrón Multicapa o MLP (Multi-Layer Perceptron) de siete entradas normalizadas y dos salidas binarias, una representa el estado normal y la otra un estado de escaneo (Fig. 4), y una red Elman con entradas y salidas iguales a las anteriores, asociando una capa de contexto a la capa oculta (Fig. 5).

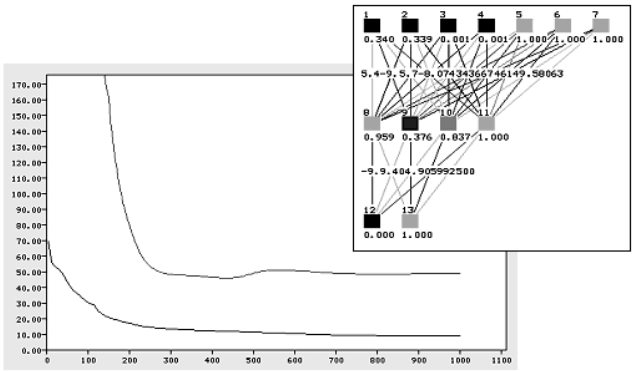

Fig. 4
RED NEURONAL MLP UTILIZADA EN EL PREPROCESADOR (DERECHA) Y SU GRÁFICA DE ERROR (IZQUIERDA)

Se utilizó la función de activación sigmóide en todas las capas tanto para la red MLP como para la Elman. La función de aprendizaje utilizada fue la llamada función de retropropagación del error (*backpropagation*) para la red MLP, y una función similar adaptada para redes parcialmente recurrentes (*Jordan and Elman Backpropagation*) para la red Elman.

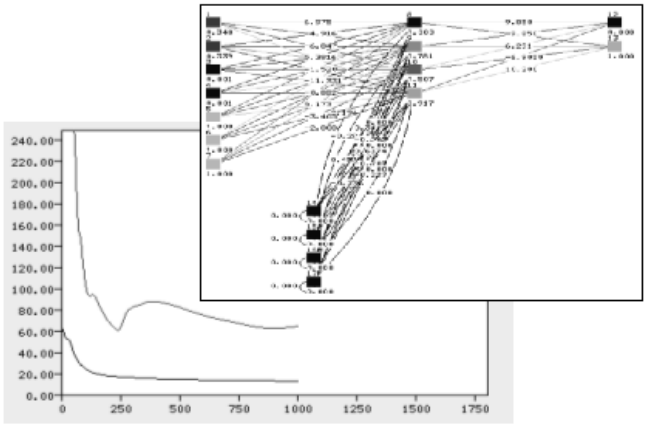

Fig. 5
RED NEURONAL ELMAN UTILIZADA EN EL PREPROCESADOR (DERECHA) Y SU GRÁFICA DE ERROR (IZQUIERDA)

En la primera fase de análisis de tráfico para identificar los patrones de cantidad y temporalidad de un ataque, se realizó un prototipo que produjo una serie de tablas donde se muestra cómo el número de paquetes de intento de conexión, así como de respuesta negativa aumentan significativamente durante un ataque, y de qué forma se diferencia este tráfico del común dentro de una hora pico o del tráfico dirigido a un servidor convencional, con el fin de minimizar en tanto sea posible el número de falsas alarmas. Un diagrama general del prototipo es mostrado en la Fig. 6.

Los parámetros que se pudieron extraer por cada dirección IP al final del estudio, basados en la cantidad de paquetes de intento de conexión y en la frecuencia con que estos eran detectados, fueron los siguientes:

A. hits_as_dst (hits as destination – hits como destino)

Número de paquetes de inicio de conexión en los cuales esta dirección IP figuraba como destino. Es decir, numero de intentos de conexión que la IP recibió. Un valor alto en este parámetro puede significar que la IP está siendo victima de un escaneo.

B. hits_as_src (hits as source – hits como origen)

Número de paquetes de inicio de conexión en los cuales esta dirección IP figuraba como origen. Es decir, numero de intentos de conexión enviados por

la IP. Un valor alto de este parámetro puede significar que el host poseedor de dicha IP está realizando un escaneo.

C. av_rcv_time (average receive time – tiempo promedio de recepción)

Tiempo promedio entre peticiones de conexión recibidas por esta IP. Esta medida dice con qué frecuencia están llegando peticiones de conexión a un equipo. Un valor bajo de este parámetro puede significar que este host está siendo escaneado.

D. av_snd_time (average send time – tiempo promedio de envío)

Tiempo promedio entre peticiones de conexión hechas por esta IP. Esta medida me dice con qué frecuencia un equipo está enviando peticiones de conexión. Un valor bajo puede indicar que el host está realizando un escaneo.

E. negative_resp (negative responses – respuestas negativas) o ack_rst_resp

Ésta medida tiene en cuenta todos los segmentos TCP, no sólo a los de inicio de conexión. Es el número de respuestas negativas que una dirección IP envía. Una respuesta negativa es el mensaje que un host genera al recibir una petición de conexión en alguno de sus puertos cuando éste está cerrado, las respuestas negativas para el protocolo TCP están dadas generalmente por el envío de un paquete con las banderas RST y ACK fijadas. Un valor alto puede indicar que esta dirección IP está siendo escaneada, ya que no es normal que un host reciba muchas peticiones en sus puertos cerrados.

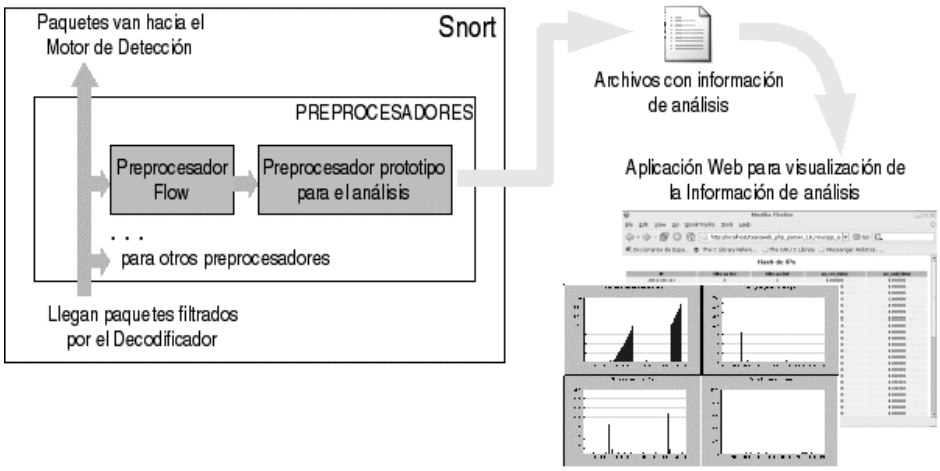

Fig. 6
DIAGRAMA MODULAR GENERAL DEL PROTOTIPO DESARROLLADO PARA EL ANÁLISIS DEL TRÁFICO

Además, por cada inicio de conexión entre dos direcciones IP se obtuvo:

F. rel_hits (relation hits – hits de relación)

Es el número de peticiones de inicio de conexión que se realizan entre dos host dados. Este parámetro sería alto en el caso de un escaneo uno a uno, ya que demostraría que hubo muchos intentos de conexión entre los dos hosts analizados.

Finalmente estos ítems vendrían a convertirse en las entradas del sistema inteligente. El uso de promedios para medir los tiempos pudiera llegar a ser motivo de controversia, ya que esta medida estadística no es confiable para datos dispersos dentro de un espacio grande de muestras, por ejemplo si son 100 muestras, y 99 de ellas están dentro de valores de 12 y 15, una última muestra que fuese digamos, 98, no sería tomada en cuenta, es decir, el 98 no afectaría el valor del promedio final que estaría entre 12 y 15. Este inconveniente fue superado estableciendo unas ventanas de análisis cuya duración es variable, configurable y depende del número de intentos de conexión, esto permite un dictamen más exacto.

La generación de los *sets* de entrenamiento para las redes neuronales se realizó de la siguiente manera: se desarrollo un prototipo que analiza el tráfico y, en tiempo real, genera archivos con la información

de ejemplo para la red neuronal (*sets* de entrenamiento). Para indicar si dicha información correspondía a tráfico de escaneo o a tráfico normal, se generaron ataques bajo condiciones controladas, indicando al prototipo que marcara dicho tráfico como tráfico de ataque o tráfico normal. De esta forma la red neuronal, por si misma, identifica las características del tráfico que se marca como normal y del que se marca como escaneo, adaptándose a las condiciones de distintos ambientes.

VII. RESULTADOS

Para determinar la calidad de la solución realizada durante el proyecto se hicieron pruebas de rendimiento y efectividad de detección de la aplicación. El host en el cual se ejecutó Snort con PortscanAI fue un Pentium 4 de 1.5GHz, 630MB de RAM y Linux Debian Sarge con kernel 2.4.27.

Las pruebas se hacen comparando el rendimiento del preprocesador **sfPortscan** (que trae Snort por defecto para detectar escaneos de puertos) con el preprocesador PortscanAI (elaborado por los autores). La configuración de sfPortscan para todas las pruebas fue: detectar en todos los protocolos (proto { all }), reportar todos los tipos de escaneo y nivel de sensibilidad bajo (sense_level { low }).

La configuración del preprocesador **PortscanAI** para todas las pruebas fue: ignorar tráfico de difusión o broadcast (ignorebc 1), límite bajo de la ventana 100, alto 1600 (analyze_thr_lower 100 analyze_thr_upper 1600), nivel de sensibilidad 0.05 (sense_level 0.05).

A. Pruebas de rendimiento

Porcentaje de paquetes perdidos

Es el porcentaje de paquetes sin analizar (*dropped)*, es decir, los paquetes que llegan al host pero no alcanzan a ser procesados debido a las limitaciones en el poder de procesamiento o a la mala utilización de éste. Este porcentaje debería ser siempre cero, pero en condiciones de mucho tráfico se incrementa. Un factor que influye en esta medida es la eficiencia con que son diseñados los algoritmos, así un algoritmo ineficiente haría perder un gran porcentaje de paquetes debido al tiempo que se toma para procesar cada uno de ellos, mientras un algoritmo eficiente no haría perder ninguno.

La medida se realizó sometiendo al host sensor a una carga constante y muy alta de tráfico, corriendo inicialmente una instalación de Snort sin PortscanAI y luego con dicho preprocesador. Los resultados obtenidos en varios intentos fueron muy variables, por esta razón los valores consignados son promedios.

En esta prueba se capturaron 109 000 paquetes en 60 segundos. Los resultados se pueden observar en la Tabla 1.

TABLA 1
PORCENTAJE DE PAQUETES PERDIDOS POR RENDIMIENTO

	Sin PortscanAI	**Con PortscanAI**
% de paquetes perdidos	0.151 %	0.896 %

Consumo de memoria

1. En condiciones iniciales: se mide cuánto aumenta el consumo de memoria de Snort cuando utiliza el preprocesador **PortscanAI** comparado con el consumo de memoria cuando se utiliza el preprocesador **sfPortscan**, al momento en que se inicia y sin someterlo a tráfico de red.
2. Con tráfico: incremento en el consumo de memoria luego de someter el sensor a tráfico generado de forma controlada.

El sensor fue sometido a 1676 paquetes TCP desde un solo origen, cada uno dirigido a un puerto distinto del sensor. Los resultados en la Tabla 2.

TABLA 2
CONSUMO DE MEMORIA DE CADA PREPROCESADOR

	sfPortscan	**PortscanAI**
Condiciones iniciales	0.3 MB	0.1 MB
Con tráfico (luego de 1676 paquetes)	0.2 MB	< 0.1 MB

B. Pruebas de efectividad

Para estas pruebas se utilizaron dos sensores: uno con sfPortscan y PortscanAI con la red MLP, y otro con sfPortscan y PortscanAI pero con la red Elman. Las alertas generadas por las dos instalaciones de sfPortscan fueron coincidentes.

Con tráfico normal
Se sometió los sensores al tráfico que circula normalmente por la red de la Universidad del Cauca y se compararon las alertas generadas por los dos, ignorando las alertas de Portsweep propias de sfPortscan.

La prueba se realizó el Miércoles 25 de enero de 2006 entre las 4:15 PM y las 5:34 PM, duró 1 hora y 18 minutos. Durante este tiempo se capturaron 2'585 956 paquetes. Los resultados se muestran en la Tabla 3.

TABLA 3
EFECTIVIDAD DE LA DETECCIÓN CON TRÁFICO NORMAL

	sfPortscan	PortscanAI	
		MLP	Elman
Escaneos detectados	2	5	1
Falsas alarmas	0	4	1
% de paquetes perdidos		0.057 %	0.045 %

Los parámetros de la tabla son: **Escaneos detectados**, indicando cuantos escaneos detectaron los sensores; **Falsas alarmas,** cuantos de dichos escaneos detectados fueron falsas alarmas (falsos positivos). Los ataques que no son considerados falsas alarmas no fueron confirmados, lo cual quiere decir que pueden ser aún falsas alarmas. Finalmente **% de paquetes perdidos**, que es el porcentaje de paquetes que no pudo analizar el sensor (*dropped)* con la red MLP y el sensor con la red Elman.

Durante un escaneo
Todas las pruebas con los tipos de escaneos presentados a continuación fueron realizadas con el software *nmap* (www.insecure.com/nmap), suprimiendo el ping que este programa suele hacer (opción -P0).

Uno a uno (TCP SYN): número de falsos positivos y falsos negativos identificados al realizar un ataque controlado de escaneo TCP SYN o semi-abierto, de un PC a otro PC. El comando para realizar el ataque fue: nmap –sS –P0 <IP victima>.

Uno a muchos (TCP SYN): similar al punto anterior, pero en este caso no se realiza el escaneo a un sólo PC victima sino a un conjunto de equipos. Se utilizaron cinco IPs como victimas. Comando: nmap –sS –P0 <rango de IPs victima>.

Uno a uno (decoy): número de falsos positivos y falsos negativos identificados al realizar un ataque controlado de escaneo con señuelos (decoys) de un host a otro host. Este tipo de escaneo hace creer a la victima que los hosts indicados por el atacante como señuelos también lo están atacando haciendo mas difícil la identificación del verdadero atacante. Los ataques muchos a uno no se contemplaron como una prueba más, debido a que su comportamiento es muy similar al escaneo con señuelos. Se utilizaron siete IPs señuelo. El comando: nmap –sS –P0 –D <conjunto de IPs señuelo> <IP victima>.

Uno a uno (TCP Connect): número de falsos positivos y falsos negativos identificados al realizar un ataque controlado de escaneo TCP Connect, de un PC a otro PC. Comando: nmap –sT –P0 <IP victima>.

Uno a muchos (TCP Connect): similar al punto anterior, pero en este caso no se realiza el escaneo a un sólo PC victima sino a un conjunto de equipos. Se utilizaron cinco IPs como victimas. Comando: nmap –sT –P0 <rango de IPs victima>.

Los resultados en la Tabla 4 muestran que el preprocesdor PortscanAI presenta un numero menor de falsos negativos.

TABLA 4
EFECTIVIDAD DE LA DETECCIÓN BAJO ESCANEO

No. DE FALSOS POSITIVOS (FALSAS ALARMAS)			
	sfPortscan	PortscanAI	
		MLP	Elman
1. Uno a uno (TCP SYN)	0	0	0
2. Uno a muchos (TCP SYN)	0	0	0
3. Uno a uno (decoy)	0	0	0
4. Uno a uno (TCP Connect)	0	0	0
5. Uno a muchos (TCP Connect)	0	0	0

No. DE FALSOS NEGATIVOS (ATAQUES SIN DETECTAR)			
	sfPortscan	PortscanAI	
		MLP	Elman
1. Uno a uno (TCP SYN)	0	0	0
2. Uno a muchos (TCP SYN)	1	0	0
3. Uno a uno (decoy)	7	0	0
4. Uno a uno (TCP Connect)	0	0	0
5. Uno a muchos (TCP Connect)	1	0	0

VIII. ANÁLISIS DE RESULTADOS

Las pruebas de rendimiento mostraron que el porcentaje de paquetes perdidos no es una medida efectiva del rendimiento de un IDS, debido a la variabilidad de sus resultados. En cuanto a consumo de memoria, se puede ver que el uso de redes neuronales con entrenamiento supervisado lo reduce en comparación con los métodos convencionales de análisis.

En las pruebas de efectividad con tráfico normal se notó que el preprocesador PortscanAI al utilizar la red MLP, es más sensible en la detección de escaneos que el preprocesador sfPortscan, presentando una tendencia mayor hacia los falsos positivos. Mientras que durante escaneos controlados, el preprocesaor PortscanAI superó a sfPortscan al no presentar falsos negativos, destacandose la falencia del sfPortscan al detectar escaneos tipo *Decoy*.

IX. CONCLUSIONES

Un componente de Inteligencia Artificial puede ser introducido con éxito al NIDS Snort incluso mejorando los resultados que se obtienen con los métodos convencionales de detección.

El uso de una Red Neuronal Artificial simplifica el trabajo del desarrollador, ya que no es necesario escribir código fuente para elaborar una. Sólo se requieren las herramientas de simulación adecuadas.

El problema principal en el plano del análisis para la implementación de una solución que integre Inteligencia Artificial es la correcta elección de las entradas del sistema, mientras que en el desarrollo, es lograr filtrar adecuadamente la información proveniente del exterior para encajar dentro de las entradas definidas.

Para un detector de escaneo de puertos es imposible desligar la cantidad y la temporalidad, ya que son

elementos complementarios, e ignorar cualquiera de ellos o malinterpretarlo generaría una gran cantidad de falsos negativos o positivos.

La ventana de tiempo en la que se analice el tráfico de red tratando de detectar una intrusión es de vital importancia, teniendo en cuenta que algunos ataques están distribuidos en tiempo, y no se hace en una misma sesión, sino en varias.

El preprocesador que usa Snort para detectar escaneos de puertos genera demasiados falsos positivos, aún en su configuración de más baja sensibilidad.

La cantidad de paquetes descartados (*dropped*) por el IDS es un factor que influye terriblemente en la capacidad de detección del preprocesador, aun más teniendo en cuenta la gran cantidad de tráfico que genera un escaneo, que puede provocar el rápido aumento dicho factor. Es por esto que el equipo donde corra el IDS (también llamado sensor) debe tener una tarjeta de red y procesador adecuados para el segmento de red al que será asignado.

Luego de consultar de forma general el estado del arte, se descubre que la propuesta planteada en este proyecto aporta nuevos resultados a la comunidad de investigadores del área. Siendo sus mayores aportes: la detección en tiempo real, los resultados comparativos expuestos en este documento, y la forma de obtener los patrones de entrenamiento de la red neuronal, directamente del tráfico analizado, sin que tengan que ser creados por el usuario.

El software de código abierto se convierte en una excelente opción para los países en vía de desarrollo, que necesitan implementar soluciones más baratas y apropiarse de nuevo conocimiento para desarrollar nuevos productos y abrir mercados.

REFERENCIAS

[1] Computer Emergency Readiness Team (2005, Diciembre 12). [Online]. Disponible: http://www.cert.org

[2] A. F. Arboleda and Ch. E. Bedón, *Sistema de Detección de Intrusos Utilizando Inteligencia Artificial*. Universidad del Cauca, Colombia: Popayán, 2006.

[3] Fyodor (2005, Diciembre 15). Network Mapping Tool [Online]. Disponible: http://www.insecure.org/nmap

[4] A. F. Arboleda and Ch. E. Bedón (2006, Enero 9). Snort diagrams for developers [Online]. Disponible: http://afrodita.unicauca.edu.co/~cbedon/snort/snort.html

[5] L. Vivas et al. (2006, Enero 9). El IDS Snort frente a un ataque de TCP reset [Online]. Disponible: http://enlaceinformatico.unicauca.edu.co/docs/snort.pdf

[6] U. Ortega. *Estado del arte: Sistemas de Detección de Intrusos*. España: Mondragon Unibertsitatea, 2004.

[7] K. Fox, R. Henning, J. Reed and R. Simonian. *A Neural Network Approach Towards Intrusion Detection*. Proc. of the 13th National Computer Security Conference, Washington, D.C., Oct. 1990, 125-134.

[8] H. Debar and B. Dorizzi. *An Application of a Recurrent Network to an Intrusion Detection System*. In IEEE, editor, International Joint Conference on Neural Networks 1992, pages 478-483.

[9] A. Cansian, E. Moreira, Carvalho and J. Bonifácio Jr. *Network Intrusion Detection using Neural Networks*. In Proceedings of the International Conference on Computational Intelligence and Multimedia Applications (ICCMA'97), 1997, pages 276-280.

[10] J. Bonifacio Jr., A. Cansian, Carvalho and E. Moreira. *Neural Networks Applied in Intrusion Detection Systems*. In Proceedings of the IEEE International Joint Conference on Neural Networks(IJCNN'98), 1998.

[11] A. Bivens, C. Palagiri, R. Smith, B. Szymanski and M. Embrechts. *Network-based Intrusion Detection using Neural Networks*. Intelligent Engineering Systems through Artificial Neural Networks, Proc. of ANNIE-2002, vol. 12, ASME Press, New York, NY, 2002, pp. 579-584

[12] M. Ramadas, S. Ostermann and B. Tjaden. *Detecting Anomalous Network Traffic with Self-Organizing Maps*. Web proceedings of the 6th International Workshop on Recent Advances in Intrusion Detection RAID, 2003.

[13] Bombonato, Fábio. *Beholder: Sistema de Detecção de Intrusos baseado em Redes Neurais.* Brasil: Universidade Católica de Brasília, 2003.

[14] Efraín Torres. *Sistema Inmunológico para la Detección de Intrusos a nivel de Protocolo HTTP.* Santafé de Bogotá: Universidad Javeriana, 2003.

[15] Jeffrey Elman. *Finding structure in time*. Cognitive Science, 1990, 14:179-211.

[16] J. P. Bigus and J. Bigus. *Constructing intelligent agents using JAVA.* New York: John Wiley & Sons, 2001.

Autores

Siler Amador Donado
Ingeniero de Sistemas
Director del Proyecto
Profesor Universidad del Cauca

Andrés Felipe Arboleda Torres
Ingeniero en Electrónica y Telecomunicaciones
Investigador y Desarrollador del Proyecto como tesis de grado.
Universidad del Cauca
2006

Charles Edward Bedón Cortázar
Ingeniero en Electrónica y Telecomunicaciones
Investigador y Desarrollador del Proyecto como tesis de grado
Universidad del Cauca
2006

Página del proyecto:
http://afrodita.unicauca.edu.co/~aarboleda/snort_ai.htm