SISTEMA DE DETECCIÓN DE INTRUSOS UTILIZANDO INTELIGENCIA ARTIFICIAL

ANDRÉS FELIPE ARBOLEDA TORRES
CHARLES EDWARD BEDÓN CORTÁZAR

Trabajo de grado presentado como requisito para optar al titulo
de Ingeniero en Electrónica y Telecomunicaciones

Director:
Ing. Esp. SILER AMADOR DONADO

UNIVERSIDAD DEL CAUCA
FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES
PROGRAMA DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES
DEPARTAMENTO DE SISTEMAS
POPAYÁN, MARZO DE 2006

TABLA DE CONTENIDO

INTRODUCCIÓN

La rápida evolución de las redes de datos hacia la convergencia con otro tipo de servicios como voz o video, ha derivado en un aumento significativo de la cantidad de información que deben manejar. De la misma manera, las necesidades empresariales han hecho propicia la proliferación de Redes Privadas Virtuales y entornos de trabajo colaborativo de alcance global. Toda esta información debe de algún modo ser asegurada contra fugas, mientras que los equipos que hospedan o gestionan servicios deben ser protegidos contra accesos no autorizados o usos indebidos. Incluso cada computador asociado corre el riesgo de ser víctima de la visita inesperada de virus tipo gusano o intrusiones directas en su sistema operativo.

En un mundo donde la información es poder, es vital implementar sistemas de protección, detección y reacción de alta disponibilidad que puedan hacer frente a las amenazas del mundo exterior de la forma más efectiva posible, y con la intervención mínima de un administrador humano. Ante este desafío se puede encontrar una gran gama de soluciones en el mercado, que poseen diferentes características, pero que adolecen de una de las más deseadas: la adaptabilidad. Una buena parte de los productos funcionan basados en reglas actualizadas cada vez que aparece un nuevo ataque o variante de alguno ya existente. Algunos usan métodos estadísticos para determinar qué tanto se aleja el comportamiento de los usuarios en determinado momento de la conducta usual; la falencia común sigue siendo la gran necesidad de administración ocasionada por las constantes actualizaciones y la generación de falsas alarmas debido a la poca adaptabilidad de los programas.

En este sentido se han planteado respuestas desde el punto de vista de la Inteligencia Artificial, que básicamente proponen que se puede identificar no sólo un ataque, sino todo un patrón de comportamiento asociado a él. Es el caso de las Redes Neuronales

que no requieren ningún tipo convencional de tratamiento algorítmico para tal detección, aparte del necesario para filtrar la información del medio y convertirla en entradas entendibles para la red (proceso también conocido como *codificación de entradas*).

A pesar de ser una tecnología con varios años de trabajo, la Inteligencia Artificial no ha penetrado de forma significativa en el mercado de la seguridad computacional, tal vez por el hecho de que la información que está en juego es demasiado importante como para confiarla al criterio sólo de una máquina, aunque esta sea monitoreada, a su vez, constantemente por un humano. Por otro lado, la capacidad de aprendizaje de un sistema inteligente, es muy limitada comparada con la humana, y no es tan simple que generalice a partir de un conjunto de casos particulares si el problema es muy complejo.

Para este trabajo se decidió utilizar una herramienta libre bajo la licencia GPL, llamada Snort™. Por varias razones, entre ellas está que es uno de los IDS (Sistemas Detectores de Intrusos) gratuitos de mayor uso en este momento a nivel mundial (la empresa gestora del proyecto, Sourcefire, fue adquirida por Checkpoint, una reconocida empresa en el campo de *apliances* para seguridad y VPNs), es de código abierto y presenta una arquitectura modular muy conveniente para desarrollar componentes similares a *plugins* para no tener que modificar elementos de su estructura interna básica. En la actualidad las comunidades de desarrolladores de software de código abierto están teniendo un auge inusitado, tanto por la misma filosofía que encierra (un ejemplo de ello es el exitoso proyecto Ubuntu con su distribución de Linux), como por la nueva forma de trabajo colaborativo que han impulsado. De la misma forma, el Software Libre 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.

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 intento al interior de la Universidad del Cauca por aplicar este tipo de tecnologías en la búsqueda de soluciones para la

problemática de la seguridad de la información en telecomunicaciones, siguiendo la filosofía del Software Libre.

No se pretende hacer un estudio exhaustivo en cada una de las tecnologías de IA, sino tomar una de ellas y presentar resultados comparativos frente a la solución ya existente. Por otro lado, existe una cantidad bastante importante de ataques y variantes, se decidió escoger sólo uno de ellos. Esto permite tener una solución más robusta en el sentido que no intenta atacar muchos problemas de una vez y probablemente no hacer algo efectivo para todos, sino demostrar con un ataque de baja complejidad, que se pueden alcanzar márgenes de confiabilidad aceptables.

En este documento se tomará una aproximación hacia el problema de la detección de intrusos desde la filosofía del Software Libre y el uso de Inteligencia Artificial, con el objetivo de generar nuevo conocimiento en el área de la Seguridad Computacional y enriquecer la base de experiencias de la comunidad de desarrolladores de nuestra región, y a nivel global.

1 MARCO TEÓRICO

1.1 INTELIGENCIA ARTIFICIAL

1.1.1 ¿Qué se quiere decir con *inteligencia*?

Cuando se habla acerca de inteligencia artificial, la pregunta que surge frecuentemente es: ¿qué se quiere decir con *inteligencia*? ¿Se quiere decir que los programas actúan como un humano, que piensan como un humano o que actúan o piensan racionalmente? Mientras que hay tantas respuestas como hay investigadores involucrados en trabajos sobre IA, basándose en lo escrito en [Bigus 2001] se puede decir lo siguiente: un programa inteligente actúa racionalmente, haciendo las cosas que nosotros haríamos, pero no necesariamente de la misma forma que nosotros las haríamos. Estos programas tal vez no pasarían el test de Turing, propuesto por Alan Turing en 1950 como una prueba para juzgar la inteligencia computacional. Pero dichos programas realizarán tareas útiles para nosotros, nos harán más productivos, nos permitirán hacer mas trabajo en menos tiempo, y ver información más interesante y menos datos sin utilidad.

1.1.2 Clasificación de la IA

Se podría decir que cada autor que ha escrito sobre este tema ha hecho una clasificación distinta de la IA, adicionalmente a esto, algunos autores no se han puesto de acuerdo en cuales tecnologías se deben consideran como de Inteligencia Artificial y cuales no.

Según [Martín & Sanz 2001], la IA se podría clasificar en Inteligencia Artificial Clásica e Inteligencia Computacional o *Soft Computing*. La IA Clásica es aquella que se realiza por medio de algoritmos convencionales siendo su más grande evolución los Sistemas Expertos.

Por su parte, las tecnologías de Inteligencia Computacional o *Soft Computing* buscan encontrar soluciones más adaptativas a los problemas, tratando de imitar a los sistemas que la naturaleza ha planteado luego de años de evolución, es decir, tratan de imitar el comportamiento o las estructuras que dotan a los seres vivos de inteligencia. Entre estas tecnologías se encuentran las Redes Neuronales Artificiales (RNA), los sistemas de Lógica Difusa y los Algoritmos Genéticos.

Según [Bigus 2001] se pueden identificar dos escuelas dentro de la IA: la escuela de Procesamiento de Símbolos y la escuela de las Redes Neuronales. En el Procesamiento de Símbolos se considera inteligencia la capacidad de reconocer situaciones o casos y que este comportamiento inteligente se puede lograr por la manipulación de símbolos.

En las Redes Neuronales o Conexionismo la inteligencia se ve reflejada en la capacidad de aprender de unos pocos ejemplos y luego generalizar y aplicar ese conocimiento a nuevas situaciones. Las redes neuronales tienen poco que ver con el procesamiento de símbolos, el cual es inspirado por la lógica matemática formal y tienen más que ver con cómo ocurre la inteligencia humana o natural.

1.1.3 Tecnologías contempladas dentro de la IA

Se excluyen de esta sección las Redes Neuronales ya que éstas son tratadas más a fondo en un ítem aparte.

1.1.3.1 Algoritmos de búsqueda

El primer acierto importante de las investigaciones en Inteligencia Artificial fue la solución de problemas utilizando técnicas de búsqueda. En esta rama de la IA los problemas son representados como estados y luego resueltos utilizando técnicas de búsqueda como simple fuerza bruta o métodos de búsqueda heurísticos más sofisticados.

Los algoritmos de búsqueda efectivos deben moverse sistemáticamente a través del espacio de estados. Los algoritmos de *fuerza bruta* se mueven ciegamente por dicho espacio mientras que los algoritmos *heurísticos* utilizan retroalimentación o información acerca del problema para direccionar la búsqueda.

Entre los algoritmos más conocidos de búsqueda por fuerza bruta se encuentran: el de búsqueda primero a lo ancho (*breadth-first search*), el de búsqueda primero a lo profundo (*depth-first search*) y el de búsqueda primero a lo profundo iterado (*iterated-deepening search*). Entre los algoritmos heurísticos se encuentran: búsqueda primero el mejor *(best-first search)*, algoritmo A*, búsqueda por satisfacción de restricciones y algoritmos genéticos.

Los *algoritmos genéticos* utilizan una metáfora con un proceso biológico. En éste método los estados del problema son representados mediante cadenas binarias llamadas cromosomas. Los cromosomas son manipulados por operadores genéticos tales como cruce y mutación. Estos algoritmos realizan la búsqueda en paralelo, donde el tamaño de la población representa el grado de paralelismo.

1.1.3.2 Sistemas expertos

Son sistemas en los que se toma el conocimiento de personas expertas en un cierto tema y se almacena en una base de conocimientos, para brindar soluciones a problemas intentando simular al experto humano.

Los sistemas expertos pueden clasificarse en dos tipos principales según la naturaleza de problemas para los que están diseñados: deterministas y estocásticos. Los sistemas expertos que tratan problemas deterministas son conocidos como *sistemas basados en reglas,* porque sacan sus conclusiones basándose en un conjunto de reglas utilizando un mecanismo de razonamiento lógico [Castillo et al. 1997].

Una característica fundamental de los Sistemas Expertos es la separación entre conocimiento (reglas, hechos) y su procesamiento. Además de tener una interfaz de usuario y un componente explicativo.

1.1.3.3 Lógica difusa

Se ha considerado de manera general que el concepto de lógica difusa apareció en 1965, en la Universidad de California en Berkeley, introducido por Lotfi A. Zadeh [Zadeh 1988].

La lógica difusa es esencialmente una lógica multivaluada que extiende a la lógica clásica. En esta última se impone a los enunciados únicamente valores falso o verdadero, mientras la lógica difusa ha modelado satisfactoriamente a una gran parte del razonamiento “natural”. El razonamiento humano utiliza valores de verdad que no necesariamente son “tan deterministas”. Por ejemplo, al calificar que “un vehículo se mueve rápido” se está tentado a graduar qué tan “rápido”, en efecto, se mueve el vehículo. Se podría decir que el vehículo se mueve “medio rápido” o “muy rápido” [Morales 2002].

1.2 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, conocida como neurona, que se encuentra interconectada con sus semejantes formando redes. Se estima que el sistema nerviosos humando esta compuesto por 100 billones de ellas.

Las Redes Neuronales Artificiales (RNA) o sistemas Conexionistas solo intentan replicar este complicado, versátil y poderoso organismo. Estas son básicamente crudos modelos electrónicos basados en la estructura neuronal del cerebro. El cerebro aprende por experiencia. Incluso los cerebros animales simples son capaces de funciones que son actualmente imposibles para los computadores. Los computadores hacen cosas de memoria bien, como guardar datos o la ejecución de matemática compleja. Pero los computadores tienen problemas al intentar reconocer simples patrones o generalizar a partir de datos iniciales.

Con esta tecnología, no se utiliza la programación tradicional sino que implica la creación de redes paralelas masivas y el entrenamiento de esas redes para solucionar problemas específicos. Este campo también utiliza conceptos de computación muy diferentes a los tradicionales, conceptos como comportamiento, reacción, auto-organización, aprendizaje, generalización y olvido [Torres 2003].

1.2.1 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, propuesta por McCullock y Pitts en 1943 (**Figura 1.1**), puede ser resumida de la siguiente manera:

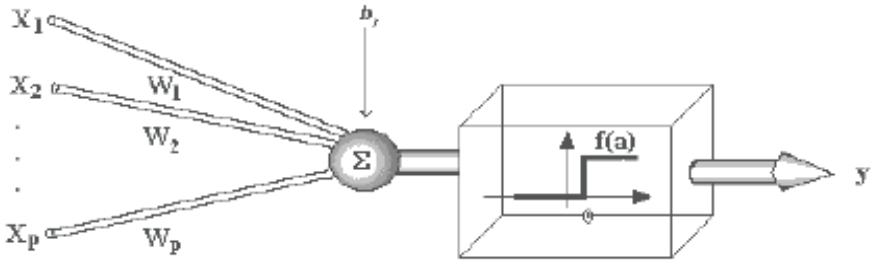


Figura 1.1. Modelo de neurona artificial de McCullock-Pitts.

- Las señales son presentadas a la entrada, representada por X_1, X_2, ... X_P
- Cada señal es multiplicada por un número, o peso, que indica su influencia en la salida de la neurona, representados por W_1, W_2, ... W_P
- Se realiza la suma ponderada de las señales que produce un nivel de actividad,
- Si este nivel de actividad excede un cierto limite (*threshold*), por el resultado de la *función de activación* (*f(a)*) la unidad produce una determinada respuesta de salida (*y*).
- En este modelo también se incluye un factor de influencia o *bias* que es aplicado externamente, representado por b_k. El bias b_k tiene el efecto de aumentar o disminuir la entrada total de activación, dependiendo si es positivo o negativo respectivamente.

1.2.1.1 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 ecuación es:

$$f(x) = \frac{1}{1 + e^{-\alpha \cdot x}}$$

Donde α es un real positivo. Cuanto más alto sea el valor de α, más bruzca será la transición de un estado a otro. En la **Figura 1.2** se ve la función para un $\alpha = 5$.

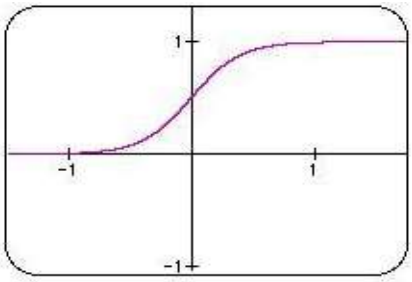


Figura 1.2. Función de activación sigmóide.

1.2.2 Topología de una red neuronal

Basándose en el patrón de conexiones (arquitectura), las RNA pueden ser agrupadas en dos categorías:

- **Redes no recurrentes (*feed-forward*)**, dónde 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 (**Figura 1.3**).

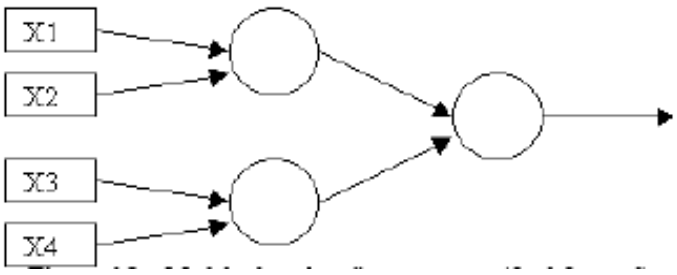


Figura 1.3. Modelo de red no recurrente.

- **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 loops debido a las conexiones de retorno (**Figura 1.4**).

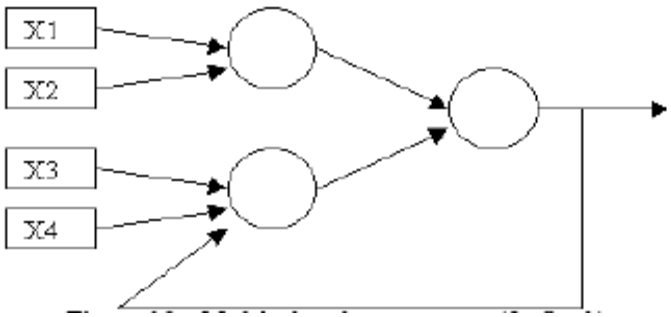


Figura 1.4. Modelo de red recurrente.

1.2.3 Aprendizaje

Las RNA aprenden a través de ejemplos. Dentro de los métodos de aprendizaje o entrenamiento, se encuentran el aprendizaje supervisado y no supervisado (también conocido como auto-aprendizaje).

1.2.3.1 Aprendizaje supervisado

Es el tipo más común, se llama supervisado porque la entrada y salida deseadas para la red son formuladas por un supervisor externo. El objetivo es ajustar los parámetros de la red, de tal forma que se abstraiga una relación entre los pares entrada-salida formulados. Se tiene un conjunto o *set* de valores de entrenamiento (o patrones), cada uno asociado con un conjunto de valores de salida deseados (*target*). Un algoritmo de entrenamiento ajusta los pesos con base en el error o diferencia entre el valor de salida obtenido de la red y el valor de salida deseado (*target*) [Bombonato 2003].

Una de las mayores ventajas de las redes neuronales es su habilidad para generalizar. Esto significa que una red entrenada podría clasificar datos como los datos de aprendizaje sin haberlos visto nunca. En aplicaciones reales los desarrolladores

normalmente tienen sólo una pequeña parte de todos los posibles patrones para la generación de una red neuronal.

La desventaja del aprendizaje supervisado es que, en ausencia de un supervisor o profesor, la red no consigue aprender nuevas estrategias para situaciones no cubiertas por los ejemplos de entrenamiento de la red. Pero las redes de aprendizaje supervisado se ejecutan mucho mas rápido que las de entrenamiento no supervisado.

1.2.3.2 Aprendizaje no supervisado

Como su nombre lo indica, no hay un supervisor para acompañar el proceso de aprendizaje. Muchos de los sistemas biológicos presentan aprendizaje no supervisado. En estos algoritmos, sólo están disponibles los patrones de entrada para la red, al contrario del supervisado cuyo conjunto de entrenamiento posee pares de entrada y salida.

Se utiliza en RNA auto-organizables, las cuales agrupan valores similares de entrada sin hacer uso de un conjunto de entrenamiento específico. Se entregan a la red una serie de valores de entrada, pero no se entregan valores de salida deseados. La red modifica sus pesos de tal forma que los valores de entrada más similares son relacionados a un mismo patrón de salida (o *cluster*). Con esto la red es capaz de producir un valor ejemplar (representativo) para cada *cluster* formado.

Algunas de las arquitecturas de redes neuronales más utilizadas serán mostradas a continuación.

1.2.4 El perceptrón

El *Perceptrón* es la forma más simple de una red neuronal usada para la clasificación de patrones *linealmente separables* (Por ejemplo: patrones que se encuentran en lados opuestos de un hiperplano). Consiste básicamente de una única neurona con pesos sinápticos ajustables y *bias*. El algoritmo usado para ajustar los parámetros libres de esta red neuronal apareció primero en un procedimiento de aprendizaje desarrollado por Rosenblatt (1958, 1962) para su modelo cerebral del perceptrón.

1.2.5 Redes perceptrón multicapa o MLP

Básicamente, la red 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 hacia el frente a través de la red, capa por capa. Estas redes neuronales son normalmente llamadas perceptrones multicapa o *Multi-Layer Perceptron* (MLP) (ver **Figura 1.5**).

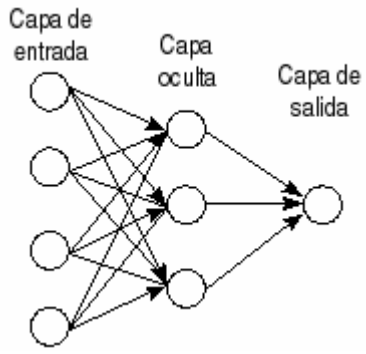


Figura 1.5. Red MLP.

La *capa de entrada* se utiliza para recibir los datos de entrada de la red. Esta capa, por consiguiente, no efectúa ningún tipo de procesamiento sirviendo apenas para recibir y almacenar el vector de entrada, o sea, el patrón a ser clasificado. El número de

neuronas de esa capa puede ser fácilmente obtenido del número de componentes del vector de entrada.

La capa de salida tiene la función de colectar y almacenar las respuestas de la red al patrón insertado en la entrada. También posee una estructura relativamente simple, siendo compuesta de tantas neuronas como clases se identifiquen en el problema.

Entre la capa de entrada y la de salida se puede tener una o mas *capas ocultas*, llamadas así por no poseer ninguna comunicación directa con los datos trabajados. La principal función de estas capas es proporcionar complejidad y no-linealidad a la red.

1.2.6 Red neuronal Elman

En 1990, el Doctor Jeffrey L. Elman en su trabajo "Finding Structure in Time" [Elman 1990] desarrolló una red neuronal basada en el trabajo previo de Jordan [Jordan 1986]. El objetivo principal era encontrar una solución con respecto al uso de redes neuronales en ambientes donde el tiempo además de las entradas era un aspecto relevante. Al tener en cuenta el tiempo, se desarrollo una arquitectura que maneja el contexto de los patrones de entrada. Así un patrón dependía del anterior y así sucesivamente. Así el tiempo ya no es representado como una parte explícita de la entrada sino que ahora el tiempo es representado por el efecto que tiene en el procesamiento. Esto significa dar al sistema procesador propiedades dinámicas que responden a secuencias temporales, básicamente una memoria. La arquitectura de una red Elman (ver **Figura 1.6**), permite considerar el uso de esta para una variedad de problemas donde interviene el procesamiento de entradas, las cuales son naturalmente presentadas en secuencia. Como por ejemplo el análisis de llamados al sistema operativo para determinar patrones de comportamiento y detectar así ataques en progreso.

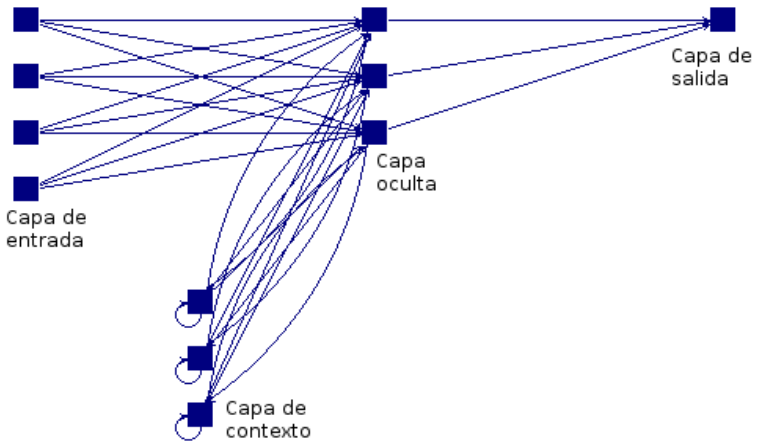


Figura 1.6. Red neuronal Elman.

1.2.7 Ventajas de las redes neuronales

Las ventajas que nos ofrecen las redes neuronales con respecto a otras tecnologías son:

- Son sistemas que aprenden lo que tienen que hacer a partir de ejemplos sin necesidad de escribir una sola línea de código.
- Son sistemas especiales para trabajar en ambientes donde la información a procesar no está muy bien definida o tiene mucho ruido.
- Son sistemas que, dadas las condiciones, pueden llegar a ofrecer soluciones de muy bajo costo y en un tiempo record de desarrollo.

1.3 SISTEMAS DE DETECCIÓN DE INTRUSOS (IDS)

La detección de intrusos es un proceso de monitoreo de eventos que ocurren en un computador o red de computadores en busca de señales de una posible intrusión, definida ésta como un intento de comprometer la confiabilidad, integridad, disponibilidad, confidencialidad de un sistema o de la información soportada por éste.

Las intrusiones son causadas por atacantes que acceden a través de Internet (redes externas), usuarios autorizados que intentan obtener privilegios adicionales para los cuales no han sido autorizados o que hacen mal uso de los que les han sido concedidos. Los Sistemas de Detección de Intrusos (IDS por sus iniciales en inglés) son productos software o hardware que automatizan este proceso de monitoreo y análisis.

1.3.1 Actividades desempeñadas por un IDS

- **Recolección de Información**: Los diferentes eventos en el entorno sirven como información para determinar sin un intruso fue capaz de obtener acceso.
- **Análisis**: Consiste en la organización y filtraje de la gran cantidad de información recolectada, esto permite seleccionar qué eventos indican que una intrusión está en curso a ya ha sucedido.
- **Respuesta**: Es el conjunto de decisiones y acciones llevadas a cabo por el sistema cuando un intruso es detectado. Se suele clasificarlas como pasivas o activas. Las primeras involucran una intervención automatizada por parte del sistema, mientras que las segundas sólo consisten en reportar las conclusiones, para que el operador del sistema decidan lo que debe hacerse.

Usualmente los IDS implementan estas características a manera de módulos, que son integrados alrededor de un motor (*engine*).

1.3.2 Tipos de IDS

Los sistemas de detección de intrusos, básicamente pueden clasificarse en dos categorías a saber: HIDS (Host IDS, o IDS de Host) que funciona en una sola máquina y monitorea sólo los eventos generados en esta, y los NIDS (Network IDS o IDS de Red), que hace lo propio pero para un segmento de red. En adición a estos tipos se suele mencionar también a los IPS (Intrusion Prevention Systems, Sistemas de

Prevención de Intrusiones), que técnicamente son una subdivisión de los anteriores. Son aquellos IDS que toman una posición proactiva ante una intrusión, y en vez de detectarla y esperar para tomar una decisión, pretenden sofocarla antes que se produzca.

1.3.2.1 Host IDS

Este tipo de IDS, normalmente se encuentra en una sola máquina, se encarga de recolectar y validar la información contenida en el host, como logs, llamadas del sistema desde el espacio de usuario al núcleo del sistema operativo, verificación de carga de la CPU y de la memoria. La ventaja de esta aproximación es que puede analizar las actividades con gran confiabilidad y precisión, determinando por ejemplo, exactamente que procesos y usuarios están involucrados en un ataque, aún más teniendo en cuenta que la información que deben manejar es relativamente poca y pueden dedicarse a la tarea de detección con mayor minuciosidad.

Este tipo de sistemas depende de la confiabilidad de las bitácoras que analiza, y que son generadas por el sistema operativo o las aplicaciones, además requiere confianza en el equipo en el que se va a instalar (no es de mucha utilidad si el equipo ha sido comprometido anteriormente). Un HIDS impacta directamente al sistema que protege en caso de falla, dado que comparte los recursos con las aplicaciones y el sistema operativo que protege, y por último, es susceptible a un ataque directo, ya que reside dentro del mismo sistema. Un ejemplo de este tipo de IDS es el Tripwire.

1.3.2.2 Network IDS

La gran mayoría de los sistemas de detección de intrusos son basados en redes. Estos IDS detectan un ataque por medio de la captura y análisis de los paquetes que viajan por un segmento de red (conformado por un hub o varios hubs en cascada) o por un

switch debidamente configurado. Un IDS basado en red puede monitorear el tráfico generado por varios hosts de la red, protegiéndolos, a la vez que genera un reporte para ese trozo de la red de la organización, para luego ser juntado con los de los demás segmentos.

Los IDS basados en red generalmente consisten en un conjunto de hosts o sensores colocados en puntos estratégicos de la red. Estas unidades o agentes monitorean el tráfico, efectuando un análisis local de la información capturada y transmitiendo los incidentes a un gestor central, como usualmente son pocos, son más fáciles de defender contra ataques. Muchos de estos sensores son configurados para ejecutarse de forma silenciosa (también llamada *stealth*), siendo mucho más complicado para los hipotéticos atacantes determinar su presencia y ser localizados, ya que este estado implica no tener IP o estar aislado de manera especial de la red.

Entre las ventajas de los NIDS se puede nombrar que de ser colocados correctamente pueden monitorear una red bastante grande y de la misma forma ser prácticamente invisibles ante posibles intrusos. Por otro lado, no significan un gran impacto para la red, ya que normalmente los sensores son dispositivos que se quedan escuchando en la red sin interferir en su funcionamiento y la poca información que genera solamente está relacionada con el gestor central.

Sin embargo también poseen desventajas: Pueden tener problemas en procesar todos los paquetes de una red o redes si están sobrecargadas de tráfico, pudiendo entonces fallar en el reconocimiento de un ataque que esté inmerso en él. En este sentido algunos vendedores han optado resolver este problema implementando el IDS en hardware especializado (*appliance*), garantizando una velocidad mayor de análisis y captura de paquetes. Además de eso, algunas investigaciones intentan disminuir el tiempo de análisis por medio del uso de técnicas probabilísticas o relacionadas con inteligencia artificial.

Otra desventaja que se puede identificar es que los IDS no se aplican a redes que usan switches. Estos aparatos subdividen la red en pequeños segmentos y proveen enlaces dedicados entre los hosts conectados a ellos, limitando el alcance de monitoreo de los IDS. Para dificultar aún más, gran parte de los switches que se encuentran en el mercado no disponen de puertos de monitoreo universales (también llamados puerto espejo o SPAN port), e incluso cuando existe, frecuentemente no deja ver todo el tráfico que atraviesa por el switch.

Además, los IDS de red no analizan la información con contenido encriptado. Este problema está en aumento, ya que cada vez más las organizaciones están usando VPNs (Virtual Private Networks, Redes Privadas Virtuales). El tráfico de datos encriptado representa una mejor seguridad, pero también puede enmascarar un posible ataque.

Por otro lado los IDS no permiten determinar si un ataque tuvo no éxito, solamente reconocen que un ataque fue iniciado. Esto significa que después de que un IDS reconoce un ataque los administradores deben manualmente investigar cada máquina afectada y determinar cuándo ocurrió para tomar las medidas del caso.

Finalmente, algunos IDS tienen dificultad tratando paquetes fragmentados. Dependiendo de la forma en que lo hagan, algunos paquetes malformados pueden hacer que el IDS se torne inestable y sea echado abajo.

Este tipo de IDS requieren emular el comportamiento de los sistemas que protege para mantener una sincronía entre su procesamiento y el de estos equipos, de no existir esta sincronía, el sistema puede ser evadido al ver cosas distintas de las que ven los equipos que protege.

Por otro lado es fundamental que posea buenas capacidades de procesamiento para evitar la pérdida de paquetes (con la consiguiente pérdida de sincronía que esto ocasiona).

1.3.2.3 Intrusion Prevention Systems (IPS)

Un IPS combina las capacidades de bloqueo de un firewall con la inspección profunda de paquetes de un IDS. Un IPS puede ser cualquier dispositivo (hardware o software) que tiene la habilidad de detectar ataques y prevenir que el ataque tenga éxito. Ahora que los firewalls pueden guardar rastro de los números de secuencia TCP y tienen la habilidad de bloquear cierto tipo de tráfico, incluso ellos pueden actuar como Sistemas de Prevención de Intrusiones (IPS).

1.3.3 Tipos de Detección

Claramente es posible diferenciar dos tipos de detección, uno es basado en **detección de uso indebido**, y el otro en **detección de anomalías**. La primera, consiste en la identificación de firmas de ataque o conjunto de eventos fijos que configuran una intrusión. La segunda clasifica como sospechosas o intrusivas aquellas desviaciones que pueda detectar sobre el comportamiento normal de un sistema.

1.3.3.1 Detección de Uso Indebido

En la detección de uso indebido se hace un análisis de la actividad del sistema, en procura de un evento o conjunto de eventos que sean similares a un patrón predefinido que describiría un ataque conocido. Los patrones que corresponden a un ataque conocido son llamados **firmas de ataque**. Una forma usual de detección especifica que cada patrón de eventos corresponde a una firma ataque unica. Sin embargo existen técnicas más sofisticadas para efectuar detección de uso indebido (llamadas basadas en estados), pueden usar una sola firma para un grupo de ataques, disminuyendo así, la cantidad de definiciones de firmas.

Como ventajas de este sistema, se puede mencionar que los detectores de uso indebido son mucho más efectivos para detectar ataques sin generar una gran cantidad de falsas alarmas, además pueden diagnosticar el uso de una herramienta o técnica de ataque específica con mayor grado de confiabilidad, ayudando así a los administradores a priorizar las medidas de corrección. Permiten que los administradores, independientemente del nivel de su conocimiento en seguridad, puedan corregir los problemas que existan en este sentido, iniciando procedimientos para enfrentar incidente, al generar una información más precisa y detallada del ataque.

Por otro lado, la detección de uso indebido solamente pude detectar los ataques que son previamente conocidos. Esto lleva a la constante necesidad de actualizaciones de las firmas.. Además, pequeñas variaciones de un ataque pueden llegar a confundir al IDS. Cuando se usa una aproximación basada en estados, se puede resolver este problema, pero los IDS comerciales no lo usan, ya que aún es experimental.

1.3.3.2 Detección de Anomalías

La detección de anomalías se hace por medio de la identificación de comportamientos anormales o poco comunes (anomalías) en una máquina o red de computadores. La teoría se da bajo la suposición de que los ataques son diferentes de una actividad normal y legítima, y puede, consecuentemente, ser detectado por el sistema que reconoce estas diferencias. Se construyen perfiles a través de los datos recolectados en un período normal de operación. Estos detectores reúnen toda esta información de eventos y utilizan una variedad de medidas para determinar cuándo las actividades monitoreadas se salen de lo normal.

Desgraciadamente, los IDS que usan el modelo de detección por anomalía, frecuentemente producen una gran cantidad de falsas alarmas, interpretando como un comportamiento anormal una conducta legítima. A pesar de estos problemas, los

investigadores afirman que los IDS por anomalía son capaces de detectar nuevas formas de ataque, a diferencia de los sistemas basados en firmas.

Entre las ventajas de los IDS por anomalía se puede nombrar la habilidad que poseen de detectar síntomas de posibles ataques, sin poseer un conocimiento específico de los detalles de estos, convirtiéndose en una herramienta relativamente adaptable.

Por otro lado producen una gran cantidad de falsas alarmas, debido a ciertos comportamientos imprevisibles de los usuarios de la propia red.

1.3.4 Problemas de identificación

Un IDS, como todo sistema, puede presentar problemas durante su operación. Estos pueden ser clasificados en: falsos positivos, falsos negativos y errores inducidos**.**

Falsos Positivos

Ocurre cuando la herramienta clasifica una acción como una posible intrusión cuando se trata de un comportamiento legítimo, que no constituye una violación de seguridad.

Falso negativo

Sucede cuando una intrusión real acontece, pero la herramienta permite que pase como si fuese una acción legítima.

Error inducido

Ocurre cuando un intruso modifica la operación del IDS para forzar la ocurrencia de falsos negativos. Esto sucede generalmente cuando el atacante logra introducirse en el host en el que opera el detector de intrusos.

1.3.5 Arquitecturas de implantación y administración de NIDS

Los sistemas de detección de intrusos de red (NIDS) se pueden administrar de manera aislada o a través de arquitecturas de 2 y 3 capas:

En un sistema de detección de intrusos que trabaja de manera aislada; los eventos se almacenan y reportan en el mismo sistema. En una arquitectura de 2 capas los NIDS son sensores que se configuran y administran desde una consola central (**Figura 1.7**).

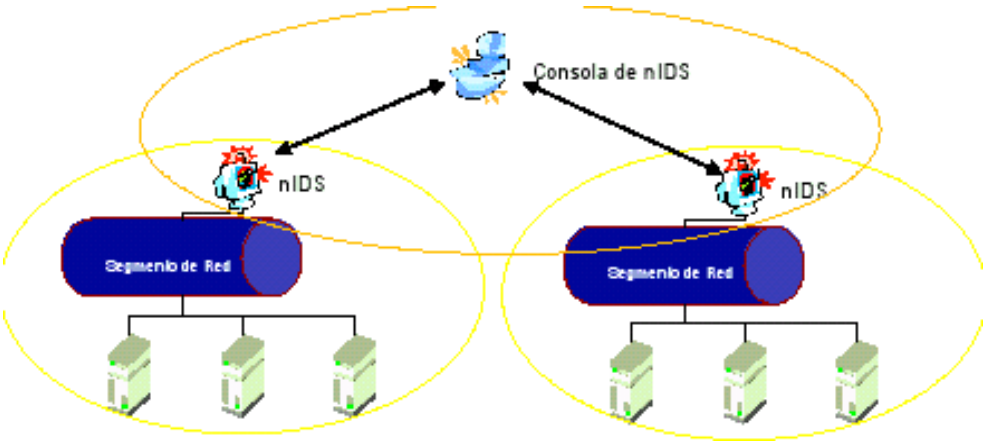


Figura 1.7. Arquitectura de 2 capas para NIDS.

Por ultimo, en una arquitectura de 3 capas; el componente intermedio (administrador), concentra las bases de datos de los sensores que maneja. Además, procesa y aplica los cambios de configuración enviados desde la consola y envía a la consola reportes de actividad de los sensores (**Figura 1.8**). En esta arquitectura es posible llevar a cabo cierto tipo de correlación en el sistema administrador, el cual concentra los datos de diversos agentes.

La selección de la arquitectura depende básicamente del tamaño de la organización, la cantidad de recursos que desea proteger, la importancia de estos recursos para la organización y los recursos que puede destinar para administrar la solución de sistema de detección de intrusos [Herrera 2003].

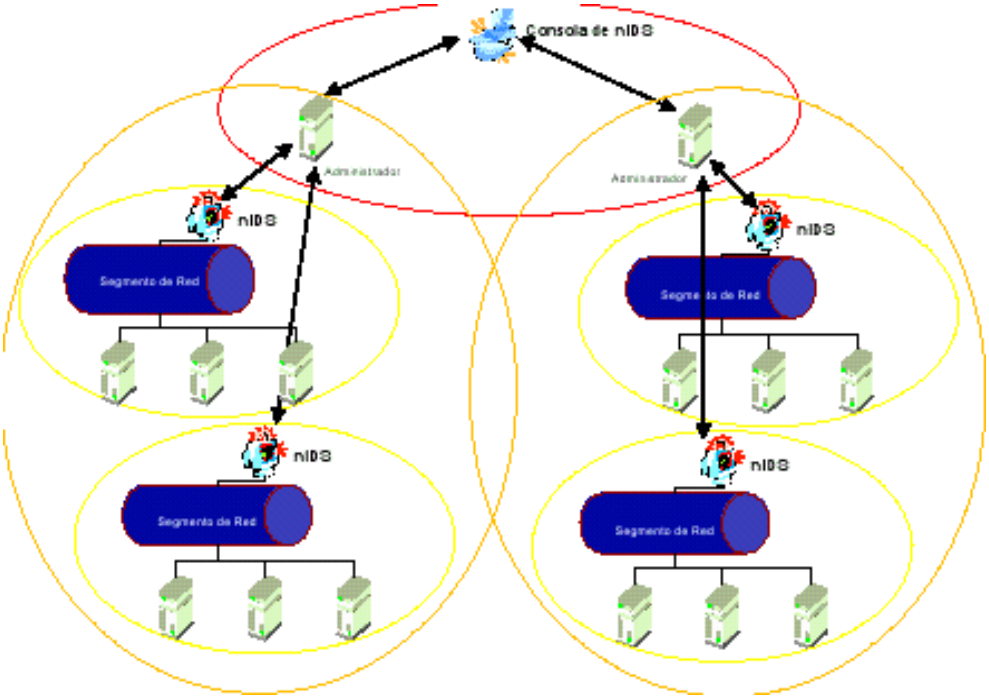


Figura 1.8. Arquitectura de 3 capas para NIDS.

1.4 ESTADO DEL ARTE DE IDS QUE UTILIZAN REDES NEURONALES

Sacado de [Ortega 2004].

Se han realizado numerosos trabajos con redes neuronales artificiales en detección de intrusos tratando de dar una alternativa a los sistemas expertos gracias a su flexibilidad y adaptación a los cambios naturales que se pueden dar en el entorno y, sobre todo, a la capacidad de detectar instancias de los ataques conocidos. La mayor deficiencia que tienen las redes neuronales es que son un modelo no descriptivo, es decir; actúan como una caja negra sin que se pueda conocer la razón de la decisión tomada.

El primer modelo de detección de intrusos basado en redes neuronales lo realizaron [Fox et al. 1990] como método para crear perfiles de comportamiento de usuarios. Al igual que en [Deb92a], 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 [Deb92b].

Más tarde, los brasileños Cansian et al., desarrollan el sistema SADI (Sistema Adaptativo de Detecção de Intrusão), que usan la red neuronal para identificar el comportamiento intrusivo de patrones capturados de una red [Can97][Bon98].

Al año siguiente, Ryan y sus compañeros desarrollaron NNID (Neural Network Intrusion Detection) para la identificación de usuarios legítimos basado en la distribución de los comandos que ejecutaban. Escogieron una arquitectura de red neuronal multicapa de tipo back-propagation de tres capas para su cometido [Rya98].

Höglund et al., del centro de investigación de Nokia, en Finlandia, propusieron el uso de dos mapas autoorganizativos. Uno de ellos, de dimensión 18x14, se utilizó como método de visualización del comportamiento de los usuarios, y el otro, un pequeño mapa que compara el comportamiento de un usuario con el perfil generado en la red anterior ayudado con medidas estadísticas [Hög98].

David Endler utilizó un perceptrón multicapa tanto para la detección de uso indebido como para la detección de anomalías a partir de datos de auditoria procedentes del BSM (Basic Security Module) de Solaris [End98].

También se han utilizado mapas autoorganizativos como técnica de visualización de logs, tras agrupar los mismos en función del puerto [Gir98], para determinar qué clusters contienen ataques.

Cannady y Mahaffey de Georgia Technical Research Institute (GTRI), dirigieron una investigación que aplicaba un modelo híbrido de perceptrón multicapa y mapas autoorganizativos para la detección de uso indebido [Cann98]. Este mismo equipo realizó otro trabajo en el 2000 utilizando múltiples mapas autoorganizativos para la detección de intrusos [Rho00].
Lippmann y Cunningham realizaron un proyecto que mejoraba el rendimiento de la detección de ataques de tipo U2R realizados mediante el uso de palabras clave [Lip99]. Una vez obtenidas las palabras clave, se usaba una red de tipo perceptrón multicapa (sin ninguna capa oculta) para la detección de ataques. Más tarde, se utilizó otra red neuronal similar para su clasificación. Una red de tipo perceptrón multicapa (sin ninguna capa oculta) mide inicialmente el número de palabras clave, proporcionando una estimación de la probabilidad posterior de un ataque en cada sesión telnet. La otra red, del mismo tipo, se utilizaba posteriormente para tratar de clasificar ataques conocidos y de esa manera facilitar el nombre de dicho ataque.

Este mismo año, Ghosh y Schwartzbard presentan un trabajo muy similar a los anteriores, pero en lugar de utilizar redes neuronales para crear perfiles del comportamiento de usuarios, utilizan la red para crear perfiles del comportamiento del software de modo que tratan de distinguir entre comportamiento de software normal y malicioso [Gho99]. Utilizan una red neuronal de tipo backpropagation (perceptrón feed-forward multicapa) con el fin de generalizar datos incompletos y posteriormente realizar la clasificación. En el 2002 Bivens y sus compañeros realizan un trabajo parecido al de Cannady en 1998. Mientras que Cannady propuso un método de detección a nivel de paquete de red, el de Bivens se basa en el método de tiempo de ventana (time-window). Este método les permite reconocer ataques multi-paquete más largos [Biv02]. También utilizan un modelo híbrido de perceptrón multicapa y mapas autoorganizativos, aunque en este caso para la detección de anomalías. Finalmente, en la Universidad de Ohio se está desarrollando 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 ha sustituido por otro que utiliza mapas autoorganizativos [Ram03].

En 2003, el brasileño Fábio Bombonato desarrolla la versión de prueba de BEHOLDER, en donde mediante una red neuronal, basándose en el motor de captura de SADI, detecta escaneos de puertos. Sin embargo, la solución que propone es muy legada y sólo queda en la versión beta, así que no alcanza una gran proyección [Bombonato 2003].

1.5 TIPOS DE ATAQUES INFORMÁTICOS COMÚNMENTE DETECTADOS POR IDSs

Dentro del proyecto de evaluación de detección de intrusos DARPA, Kendall [Ken98] desarrolló una base de datos de ataques similar, que se puede encontrar en los conjuntos de datos de evaluación de detección de intrusos DARPA [DAR04]. En esta base de datos, utilizada actualmente como elemento evaluador y comparativo de los sistemas de detección desarrollados por los investigadores, los ataques se clasifican en cuatro grupos principales, utilizando como criterio el *tipo de ataque*.

1.5.1 Denegación de Servicio (DoS)

Estos ataques tratan de *detener el funcionamiento de una red, máquina o proceso; o sino denegar el uso de los recursos o servicios a usuarios autorizados* [Mar01]. Hay dos tipos de ataques DoS; por un lado ataques de sistema operativo, los cuales tratan de explotar los fallos en determinados sistemas operativos y pueden evitarse aplicando los respectivos parches; y ataques de red, que explotan limitaciones inherentes de los protocolos e infraestructuras de red.

1.5.2 Indagación o exploración (probing - *portscan*)

En cierta forma se puede decir que la exploración de vulnerabilidades empieza desde la misma elaboración del perfil de la víctima, ya sea ésta una persona o una organización. Este paso se conoce como *footprinting.*

Luego, se procede a un escaneo de las redes tratando de identificar direcciones IP válidas y recoger información acerca de ellas (servicios que ofrecen, sistemas operativos que usan). A menudo, esta información provee al atacante una lista de vulnerabilidades potenciales que podrían ser utilizadas para llevar a cabo ataques a los servicios y las máquinas escogidas. Estos ataques son los más frecuentes, y a menudo son precursores de otros ataques. Los ataques de escaneo pueden obtener:

- La topología de la red objetivo.
- Los tipos de tráfico permitidos en un firewall.
- Los hosts activos en la red.
- Cuales sistemas operativos están corriendo esos hosts.
- El software de servidor que están corriendo y/o las versiones de los distintos programas detectados.

El **escaneo de puertos (*portscan*)** es seguramente la técnica de hacking más usada en el mundo, pues en cualquier ataque -bien se trate de una intrusión ilegal o de una auditoría legal- medianamente bien planificado, uno de los primeros pasos ha de ser obligatoriamente el escaneo metódico de los puertos de la máquina en cuestión. Lo malo es que la mayoría de la gente no sabe absolutamente nada sobre cómo funciona un escaneo de puertos. Se puede identificar un gran número de formas para cumplir con este cometido.

1.5.2.1 TCP Connect()

Esta técnica es quizá la más común en cualquier software de escaneo de puertos. Consiste en usar la llamada connect() de TCP para intentar establecer una conexión con cada uno de los puertos del host a escanear. Si la conexión se establece, el puerto está abierto (escuchando conexiones); en caso de recibir un aviso de cierre de conexión (RST), el puerto estará cerrado; y en caso de no recibir respuesta, se deduce que el puerto está silencioso. Este tipo de escaneo es extremadamente rápido, pues puede realizarse de forma paralela para distintos puertos mediante el uso de varios sockets. Además, es un escaneo fácil de implementar. Su principal desventaja es que es llamativo en exceso, pues intenta establecer cientos o miles de conexiones en un margen de pocos segundos. Además, al realizarse intentos completos de conexión, cualquier sistema guardará registros.

Comportamiento del escaneo:
Las siguientes lineas simbolizan los paquetes TCP intercambiados entre el host local (atacante) y el host remoto (víctima). Cada flecha muestra la dirección del paquete y las banderas activas del paquete estan entre corchetes cuadrados.

host local ---[SYN]---> [O] Puerto TCP abierto en el host remoto
host local <---[SYN/ACK]--- [O] Puerto TCP abierto en el host remoto
host local ---[ACK]---> [O] Puerto TCP abierto en el host remoto
host local ---[SYN]---> [X] Puerto TCP cerrado en el host remoto
host local <---[RST]--- [X] Puerto TCP cerrado en el host remoto
host local ---[SYN]---> [~] Puerto TCP silencioso en el host remoto -SIN RESPUESTA

En nmap se puede invocar un escaneo TCP connect() mediante el comando:

nmap -P0 -sT xxx.xxx.xxx.xxx

1.5.2.2 TCP SYN

Esta técnica, también conocida como Half-open scan (es el escaneo medio abierto por excelencia), es parecida a la anterior con la importante salvedad de no establecer completamente las conexiones. En primer lugar, se envía un paquete SYN que finge intentar establecer una conexión y se espera la respuesta. Si llega un paquete SYN/ACK significa que el puerto está abierto; si llega un paquete RST, el puerto está cerrado; y si no se recibe respuesta se asume que está silencioso. En el caso de que el puerto esté abierto y recibamos el paquete SYN/ACK (es decir, están completos dos de los tres pasos del saludo en tres tiempos), nosotros no responderemos con un paquete ACK como sería lo esperado, sino que mandaremos un paquete RST. ¿Para qué? Pues precisamente para evitar que se complete el inicio de conexión y, por tanto, evitar que el sistema registre el suceso como un intento de conexión. En sistemas sin protección específica de cortafuegos o IDS, este escaneo suele pasar desapercibido. Una característica importante de este escaneo es que requiere elevados privilegios en el sistema para poder lanzarlo, debido a que este tipo de paquetes usan sockets TCP raw. Por tanto, solo el root puede lanzar escaneos TCP SYN. La principal ventaja de este tipo de escaneo es que suele ser bastante discreto y ofrece unos resultados bastante buenos. Entre sus desventajas se encuentra el que es algo lento de realizar, y que un sistema con un firewall o un IDS (aunque algunos muy básicos no) lo detectará e identificará como escaneo de puertos sin ninguna duda.

Comportamiento del escaneo:

host local ---[SYN]---> [O] Puerto TCP abierto en el host remoto
host local <---[SYN/ACK]--- [O] Puerto TCP abierto en el host remoto
host local ---[RST]---> [O] Puerto TCP abierto en el host remoto
host local ---[SYN]---> [X] Puerto TCP cerrado en el host remoto
host local <---[RST]--- [X] Puerto TCP cerrado en el host remoto
host local ---[SYN]---> [~] Puerto TCP silencioso en el host remoto -SIN RESPUESTA

En nmap podemos invocar un escaneo TCP SYN mediante el comando:

nmap -vv -P0 -sS xxx.xxx.xxx.xxx

1.5.2.3 TCP FIN

El escaneo TCP FIN, también conocido como Stealth scan (se trata del escaneo silencioso más conocido), es uno de los más discretos que podemos encontrar dentro de las técnicas convencionales. Se apoya en una particularidad de los estándares internacionales de TCP/IP. A la hora de realizar el escaneo, se envía un paquete FIN al puerto del host destino que queremos escanear. Los estándares de TCP/IP dicen que al recibir un paquete FIN en un puerto cerrado, se ha de responder con un paquete RST. Así pues, si se recibe RST por respuesta, el puerto está cerrado, y en caso de no recibir respuesta (se ignora el paquete FIN) el puerto puede encontrarse abierto o silencioso. Esto supone uno de los principales inconvenientes del escaneo TCP FIN, y es que los puertos que nos figuran como abiertos, pueden estar en realidad en estado silencioso (puesto que un puerto silencioso por definición ignora cualquier paquete recibido). Así pues, este tipo de escaneo no obtiene unos resultados fiables, y ese es su talón de Aquiles. Otra gran desventaja de este sistema de escaneo viene de los sistemas Windows, en los cuales un puerto cerrado ignora los paquetes FIN, por lo que escanear un sistema de este tipo con SYN FIN nos generará una enorme lista de puertos abiertos, aunque realmente estén cerrados o silenciosos. Como ventaja, tenemos el que estos escaneo pasan desapercibidos en la gran mayoría de los firewalls, al no intentar establecer ninguna conexión. Un IDS bien configurado, lo detectará.

Comportamiento del escaneo:

host local ---[FIN]---> [O] Puerto TCP abierto en el host remoto -SIN RESPUESTA
host local ---[FIN]---> [X] Puerto TCP cerrado en el host remoto
host local <---[RST]--- [X] Puerto TCP cerrado en el host remoto
host local ---[FIN]---> [~] Puerto TCP silencioso en el host remoto -SIN RESPUESTA

En nmap podemos invocar un escaneo TCP FIN mediante el comando:

nmap -vv -P0 -sF xxx.xxx.xxx.xxx

1.5.2.4 UDP scan

Esta técnica, frente a las demás técnicas orientadas a TCP, está orientada al protocolo UDP y sus puertos. Aunque a priori parezca que los puertos UDP no son muy interesantes, servicios como el rpcbind de Solaris, TFTP, SNMP, NFS... usan todos ellos UDP como protocolo de transferencia. El sistema de escaneo consiste en mandar un paquete UDP vacío (0 bytes de datos) al puerto que deseamos escanear. Si el puerto está cerrado, el sistema responderá con un paquete ICMP de tipo 3 (destino inalcanzable). En caso de no responder, el puerto puede estar abierto o silencioso. Este sistema puede presentar un grave problema de carencia de velocidad según en qué sistemas, y es que en el RFC #1812- Requirements for IP version 4 routers (ftp://ftp.rfc-editor.org/in-notes/rfc1812.txt) se recomienda limitar la capacidad de generación de mensajes ICMP de error. En sistemas Linux (consultar el fichero /ipv4/icmp.h de las fuentes del kernel) esta limitación está fijada en unos 20 mensajes por segundo. Sistemas como Solaris son más estrictos y tienen la limitación fijada en 2 por segundo. Los sistemas Windows no tienen ninguna limitación prefijada. Un escaneo UDP a un sistema Windows resulta extremadamente rápido como consecuencia de ello.

Comportamiento del escaneo:

host local ---{UDP}---> {O} Puerto UDP abierto en el host remoto -SIN RESPUESTA
host local ---{UDP}---> {X} Puerto UDP cerrado en el host remoto
host local <---|ICMP #3|--- {X} Puerto UDP cerrado en el host remoto
host local ---{UDP}---> {~} Puerto UDP silencioso en el host remoto -SIN RESPUESTA

En nmap se puede invocar un escaneo UDP mediante el comando:

nmap -vv -P0 -sU xxx.xxx.xxx.xxx

1.5.2.5 ACK scan

La mayoría de las técnicas de escaneo nos permiten identificar con exactitud los puertos abiertos o cerrados, pero generalmente los puertos silenciosos no se pueden identificar con claridad. El escaneo ACK está destinado a identificar de forma precisa cuándo un puerto se encuentra en estado silencioso. Esta técnica es usada también para poder escanear hosts que estén detrás de un firewall que bloquee los intentos de conexión (paquetes SYN). Su funcionamiento se basa en el envío de paquetes ACK con números de secuencia y confirmación aleatorios. Cuando reciba el paquete, si el puerto se encuentra abierto, responderá con un paquete RST, pues no identificará la conexión como suya; si el puerto está cerrado responderá con un paquete RST, pero si no se obtiene respuesta (obviamente primero debemos asegurarnos que el host está en línea) podemos identificar claramente el puerto como filtrado (puerto silencioso). Normalmente el escaneo ACK se realiza como apoyo a un escaneo anterior, para determinar los puertos silenciosos y poder identificar mediante una combinación de técnicas el estado real de todos ellos. Por ejemplo, ante un host con un firewall que bloquee intentos de conexión (SYN), podemos realizar un FIN scan para determinar los puertos cerrados, y después un ACK scan para determinar qué puertos están abiertos y cuáles silenciosos. Esta técnica también es usada como variante del ping (ICMP echo) de toda la vida, para saber si un host está activo (recibiremos respuesta RST) o no (cuando no hay respuesta o la respuesta es destino inalcanzable).

Comportamiento del escaneo:

host local ---[ACK]---> [O] Puerto TCP abierto en el host remoto
host local <---[RST]--- [O] Puerto TCP abierto en el host remoto
host local ---[ACK]---> [X] Puerto TCP cerrado en el host remoto
host local <---[RST]--- [X] Puerto TCP cerrado en el host remoto
host local ---[ACK]---> [~] Puerto TCP silencioso en el host remoto -SIN RESPUESTA

En nmap podemos invocar un escaneo ACK mediante el comando:

nmap -vv -sA xxx.xxx.xxx.xxx

1.5.2.6 Null scan

Este escaneo tiene muchos puntos en común con el escaneo FIN. Su funcionamiento base es el mismo: enviamos un paquete malformado (en este caso se trata de un paquete TCP con todos los flags desactivados) y esperamos la respuesta. En caso de que el puerto destino esté cerrado, nos responderá con un paquete RST; y en caso de no recibir nada (nuestro paquete es ignorado), se trata de un puerto abierto o silencioso. La ventaja frente al escaneo FIN radica en que ciertos firewalls vigilan los paquetes de finalización de conexión además de los de establecimiento, de forma que el escaneo nulo podrá realizarse allí dónde el FIN no sería posible. El resto de ventajas y desventajas son las mismas que en el escaneo FIN.

Comportamiento del escaneo:
host local ---[ ]---> [O] Puerto TCP abierto en el host remoto -SIN RESPUESTA
host local ---[ ]---> [X] Puerto TCP cerrado en el host remoto
host local <---[RST]--- [X] Puerto TCP cerrado en el host remoto
host local ---[ ]---> [~] Puerto TCP silencioso en el host remoto -SIN RESPUESTA

En nmap podemos invocar un escaneo Null mediante el comando:
nmap -vv -P0 -sN xxx.xxx.xxx.xxx

1.5.2.7 Xmas scan

El escaneo Xmas se basa también en el principio de la respuesta RST por parte de un puerto cerrado al recibir un paquete incorrecto (como el escaneo FIN). En el caso del escaneo Xmas, se trata de un paquete con los flags FIN, URG y PSH activados (aunque ciertas implementaciones activan FIN, URG, PSH, ACK y SYN e incluso algunas activan todos los flags). Podría decirse que es lo contrario del escaneo Null, pero logrando el mismo efecto. Al igual que el escaneo Null, se usa bajo ciertas circunstancias en las que el escaneo FIN no es posible; y también comparte con éstos sus particularidades.

Comportamiento del escaneo:

host local ---[xmas]---> [O] Puerto TCP abierto en el host remoto -SIN RESPUESTA
host local ---[xmas]---> [X] Puerto TCP cerrado en el host remoto
host local <---[RST]--- [X] Puerto TCP cerrado en el host remoto
host local ---[xmas]---> [~] Puerto TCP silencioso en el host remoto -SIN RESPUESTA

En nmap podemos invocar un escaneo Xmas mediante el comando:

nmap -vv -P0 -sX xxx.xxx.xxx.xxx

1.5.2.8 SYN/ACK scan

Este tipo de escaneo tiene una base parecida a los anteriormente citados FIN, Null y Xmas, pero con la sustancial diferencia de que en este caso los paquetes malformados fingen ser un error en la transacción de una conexión legítima. Mediante esta técnica, se envía un paquete SYN/ACK al puerto que deseamos escanear en el host remoto. Si el puerto se encuentra cerrado, nos responderá con un paquete RST. En caso de estar abierto o silencioso, simplemente ignorará el paquete y no obtendremos respuesta. Como ventaja, este tipo de escaneo evade la mayoría de firewalls e IDS sencillos, pero

comparte con los escaneos anteriormente citados sus problemas, principalmente la falta de fiabilidad a la hora de determinar los puertos abiertos o silenciosos.

Comportamiento del escaneo:

host local ---[SYN/ACK]---> [O] Puerto TCP abierto en el host remoto -SIN RESPUESTA
host local ---[SYN/ACK]---> [X] Puerto TCP cerrado en el host remoto
host local <---[RST]--- [X] Puerto TCP cerrado en el host remoto
host local ---[SYN/ACK]---> [~] Puerto TCP el host remoto -SIN RESPUESTA

En nmap no se encuentra implementado este tipo de escaneo.

1.5.2.9 Decoy scan

Esta técnica se utiliza acompañada con el TCP SYN. Trata de engañar a la víctima y a los posibles sistemas de detección asociados a ella haciendoles creer que muchos computadores están realizando el ataque cuando es sólo uno el que lo está haciendo. El uso de señuelos o Decoys puede hacer mas dificil la identificación del verdadero atacante, incluso algunos sistemas IDS pueden ser engañados si se utiliza un abundante número de señuelos.

En nmap podemos invocar un escaneo Decoy mediante el comando:

nmap -vv -P0 -sS -D xxx.xxx.xxx.xxx,xxx.xxx.xxx.xxx,... xxx.xxx.xxx.xxx

1.5.2.10 Ping sweep

Un ping sweep (también llamado barrido de ping) no es en realidad una técnica de escaneo de puertos, sino más bien una técnica de escaneo de hosts. Es el momento de hablar de una opción de nmap que se ha incluido en el ejemplo de todos los escaneos

de los que se ha hablado hasta ahora. Efectivamente, mediante esta opción se logra que, antes de realizar el escaneo de puertos propiamente dicho, el software no compruebe si el host está activo. Es una técnica muy común el denegar, vía firewall, la salida de paquetes ICMP echo reply. Así, al realizar un ping a un host, éste no responde y puede parecer que esté inactivo. Ahora imagine que el objetivo del escaneo va a ser toda una red (por ejemplo 192.168.0.0/24), en lugar de un único host. Lo primero que será de interés es saber qué hosts están activos, pues si se escanea toda la red con la opción -P0, perderemos mucho tiempo mandando paquetes y esperando la respuesta de equipos que en realidad están inactivos. Pero cabe la posibilidad de que algunos equipos intenten hacer creer que están inactivos cuando en realidad no lo están, y aquí es donde entran las diversas técnicas de ping sweep, que sin embargo no serán descritas aquí por estar fuera del alcance del proyecto.

1.5.3 R2L (Remote to Local)

Cuando un atacante que no dispone de cuenta alguna en una máquina, logra acceder (tanto como usuario o como root) a dicha máquina. En la mayoría de los ataques R2L, el atacante entra en el sistema informático a través de Internet.

1.5.4 U2R (User to Root)

Este tipo de ataque se produce cuando un atacante que dispone de una cuenta en un sistema informático es capaz de elevar sus privilegios explotando vulnerabilidades en los mismos, un agujero en el sistema operativo o en un programa instalado en el sistema. Este tipo de ataques son detectados por los HIDS.

Existen muchos tipos de ataques informáticos, sin embargo, sólo se expondrá los ataques que son detectados comúnmente por los IDS. Estos son básicamente tres: escaneos al sistema, denegación de servicio (DoS – Denial of Service) y penetración en

los sistemas. Estos ataques pueden ser lanzados localmente, en la maquina atacada, o remotamente, utilizando acceso a la víctima a través de la red. El operador de un IDS debe entender las diferencias entre estos tipos de ataques, ya que cada uno de estos requiere un tipo diferente de respuesta.

1.6 EL SISTEMA DE DETECCIÓN DE INTRUSOS SNORT

Snort (www.snort.org) es un IDS y desde su versión 2.3.0 también tiene funciones de IPS gracias a la incorporación de la modalidad *Snort inline*. El iniciador del proyecto fue Martin Roesch, quien desarrolló la primera versión completamente en lenguaje C. Snort es gratuito y de código abierto bajo la licencia GPL de GNU. Actualmente el proyecto está respaldado por la empresa SourceFire fundada por el mismo Roesch, la cual fue comprada en el 2005 por la multinacional Checkpoint.

Snort se apoya en un conjunto de reglas donde se tipifican patrones de ataques de los que se tiene clara certeza acerca de su funcionamiento, sobre estas reglas se basa para alertar en el caso de que algún evento que encaje dentro de esos patrones se produzca en la red que está monitoreando. Dichas reglas pueden ser construidas por el usuario lo cual le da un perfil personalizado a la protección de la red.

1.6.1 Funcionamiento de Snort

Snort en su versión actual (2.4.3) tiene cuatro modos de funcionamiento a saber, **Sniffer:** sólo monitorea el tráfico de la red y lo muestra en pantalla. **Packet Logger:** Guarda los paquetes en archivos planos o binarios. **NIDS:** De acuerdo a ciertos parámetros de configuración, alerta y/o registra los eventos que deben ser verificados posteriormente por el administrador. **Inline:** Puede impedir el paso de paquetes según los parámetros de configuración.

En el modo IDS, cuando Snort recibe un paquete lo hace pasar por diferentes módulos encargados de funciones específicas de decodificación, análisis y generación de reportes. En la **Figura 1.9** se muestra un diagrama descriptivo. Las funciones de cada módulo se describen a continuación:

Decodificador: Encaja los paquetes capturados dentro de estructuras de datos predefinidas e identifica los protocolos de capa de enlace. Luego, sube de nivel, decodifica IP y seguidamente TCP o UDP según sea el caso para obtener información como puertos y direcciones. Snort alertará si encuentra encabezados malformados, opciones TCP de longitudes inusuales o poco usadas.

Preprocesadores: Estos son módulos software que dan gran libertad a los programadores para realizar diversos procesamientos con los paquetes capturados. Existen preprocesadores que realizan tareas como reensamblar paquetes IP, segmentos TCP, realizar seguimiento detallado de conexiones establecidas, detectar ataques a través de la inspección de HTTP y muchas otras funciones útiles.

Archivos de Reglas: Archivos de texto plano donde se encuentra un listado de reglas en una sintaxis definida. Dentro de esta sintaxis se especifica el protocolo, las direcciones y los plug-ins de detección, entre otras cosas. Estos archivos de reglas se actualizan periódicamente de forma similar a como se hace con los archivos de definiciones de un antivirus.

Plug-ins de Detección: Módulos que son referenciados desde su definición en el *archivo de reglas* y sirven para identificar patrones cuando alguna de ellas es evaluada.

Motor de Detección: Haciendo uso de los plug-ins de detección, evalúa los paquetes con las reglas que se han cargado en memoria desde la inicialización del IDS.

Plug-ins de Salida: Formatean las notificaciones (alertas, logs) para que puedan ser guardadas con diferentes mecanismos (archivos binarios, archivos en texto plano, bases de datos, etc.)

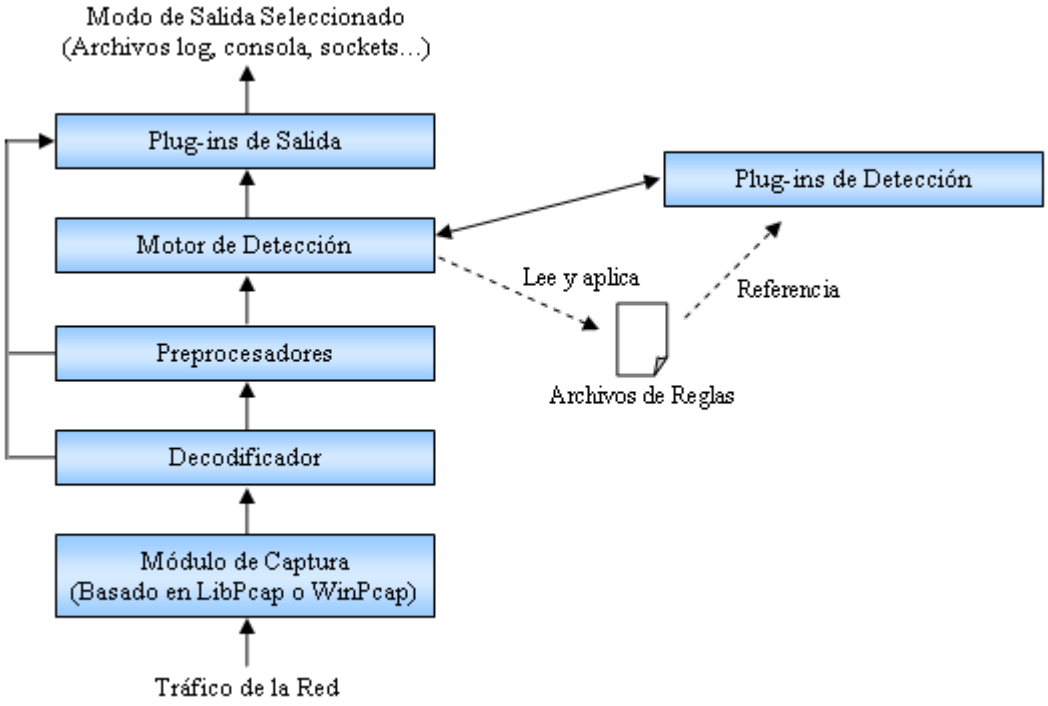


Figura 1.9. Módulos componentes de NIDS Snort.

El crecimiento y optimización de Snort se ha facilitado en gran parte gracias a esta arquitectura modular, la cual ha permitido que los miembros de la comunidad de desarrolladores añadan nuevos *preprocesadores* o *plug-ins* sin tener que preocuparse por el resto del sistema.

1.6.2 Snort frente a los ataques de indagación (escaneos)

Snort ha utilizado diferentes técnicas y módulos para la detección de este tipo de ataques pasando por el preprocesador *portscan detector* en la versión 2.1, que fue reemplazado por el preprocesador *flow-portscan* en la versión 2.2, sólo lograban

detectar ataques uno a uno y uno a muchos. Desde la versión 2.3 el preprocesador por defecto para la detección de indagaciones es el *sfPortscan*, que tiene la capacidad de detectar ataques: muchos a uno y muchos a muchos, además de reportar Decoy scans (de una forma bastante desacertada).

Pero la detección de estos ataques no se ha intentado únicamente con preprocesadores, también hay un conjunto de reglas agrupadas en el archivo *scan.rules*, que han sido actualizadas en varias oportunidades, pero no han llegado a un nivel de depuración suficiente y generan gran cantidad de falsas alarmas.

2 CONSIDERACIONES DEL PROBLEMA

2.1 INTEGRACIÓN DE TECNOLOGÍAS DE IA EN LA DETECCIÓN DE INTRUSOS

La gran cantidad de falsas alarmas (falsos positivos) que generan los productos IDS disponibles en la actualidad y la evidencia de ataques que ocurrieron sin que los hubieran detectado (falsos negativos), dejan ver la necesidad de utilizar un nuevo enfoque a la hora de procesar la información que capturan los sensores IDS.

Igualmente, la gran cantidad de firmas de ataque que deben ser actualizadas periódicamente para la correcta detección de las nuevas amenazas informáticas, aunque sean pequeñas variaciones de las antiguas, por parte de los IDS basados en firmas (cuyo uso es el más extendido en el mercado actual). Muestra la necesidad de utilizar tecnologías inteligentes, capaces de adaptarse a los pequeños cambios en los patrones de comportamiento de los usuarios y de los ataques, que disminuyan la dependencia de los detectores de intrusos a la constante actualización de sus firmas.

Las técnicas de Inteligencia Artificial podrían entran a sustituir en parte la suspicacia y conocimiento que los administradores tienen de la red, para tomar decisiones con una mayor autonomía y adaptabilidad que las presentes en los sistemas actuales. Sin embargo están lejos de reemplazar el factor humano y se han limitado a resolver pequeños problemas bien acotados (aquí es donde han demostrado su efectividad).

Hay actividades que inevitablemente requieren la presencia de un ser humano altamente capacitado. ¿O acaso se puede imaginar a un IDS llamando por teléfono al administrador de una red en China, para solicitar mayor información sobre un incidente de seguridad que, presuntamente, se originó desde ahí?

Es notable el incremento en la investigación sobre nuevas tecnologías (entre ellas las de IA) para el desarrollo de sistemas de detección de intrusos. Desafortunadamente la implementación de soluciones en un ambiente real ha sido muy pobre, quedando limitados estos nuevos aportes a proyectos de investigación académica en las universidades. Mientras tanto las organizaciones siguen adquiriendo los mismos sistemas comerciales basados en firmas, de alto costo y que han probado ser ineficientes y poco efectivos ante nuevas técnicas de ataque.

Esta situación permite deducir que la investigación en nuevas tecnologías de detección de intrusos es aun joven y que se requiere más trabajo en estas áreas, o que las nuevas tecnologías para IDS como la IA no son efectivas en este campo.

Se deja entonces la siguiente pregunta, que se espera resolver con los resultados de este proyecto de investigación: ¿es verdad que las tecnologías adaptativas de IA mejorarán de alguna forma a los IDS?

2.2 ¿POR QUÉ UTILIZAR EL NIDS SNORT, PARA LA REALIZACIÓN DEL PROTOTIPO?

Snort es un NIDS de código abierto y es software libre bajo los términos de la licencia GPL de GNU, lo cual elimina los costos de licencia por su utilización o modificación. Existen otros IDS de código abierto entre los cuales se encuentran:

- Tripwire – Este es un HIDS con dos versiones: código abierto y aplicación comercial. La versión de código abierto está alojada en SourceForge.
- AIDE – *Advanced Intrusion Detection Enviroment*. Es un reemplazo gratuito para Tripwire, ofrece las mismas funcionalidades de éste último y un poco mas.
- Prelude – Es un Sistema de Detección de Intrusos Híbrido, permite detectar rastros de actividad maliciosa desde diferentes sensores, entre ellos el Snort.

Sin embargo, Snort ofrece características que lo hacen más apropiado para el trabajo en un proyecto académico, como son:

- Su comunidad de desarrolladores es activa, está dispuesta a resolver dudas y acoger nuevos proyectos que enriquezcan su base de conocimientos y desarrollos.
- Facilita la adición de nuevos desarrollos debido a su arquitectura modular, como se pudo ver en el marco teórico.
- Adicionalmente, es el NIDS de código abierto con mayor reconocimiento como se puede ver en el artículo de [Noonan 2005] cuando menciona: "*Snort is the most popular open source IDS available*".

Como producto competitivo ante sus semejantes comerciales, Snort es claramente superado por otros productos como VisualLookout y Sentaurus (Ver **Tabla 2.1**) en aspectos como: documentación disponible, facilidad de instalación y facilidad de administración (sacado de [Noonan 2005]).

	Snort 2.2	**Sentaurus 5.2.1**	**VisualLookout 5.0g**
Documentación disponible (10%)	3	3	7
Facilidad de instalación (10%)	3	10	10
Funcionalidad (30%)	8	9	3
Rendimiento (25%)	8	7	6
Facilidad de administración (25%)	2	7	5
Puntuación total:	**5.5**	**7.5**	**5.3**
Puntuación: 1 = Virtualmente inoperable o inexistente, 2 = Promedio, se realiza adecuadamente, 10 = Excepcional.			

Tabla 2.1. Comparación de NIDS.

A pesar de las falencias de Snort, existen productos comerciales basados en éste, como el *Sourcefire IS-2000* ganador del premio *Best Buy* de la revista SC Magazine de Estados Unidos en el 2005 (Ver [SCMagazine 2005]). Este producto utiliza Snort como

su motor de detección y superó a las soluciones ofrecidas por empresas como *Symantech, McAfee* e *Internet Security Systems*.

2.3 ¿QUÉ TECNOLOGÍA DE IA UTILIZAR?

Al hablar de inteligencia artificial, se habla de un gran conjunto de tecnologías. Al iniciar este proyecto no se tenía idea de cuales eran las que se contemplaban dentro de dicho conjunto, por esta razón, se estudiaron sólo las tecnologías que se consideraban más populares dentro de esta rama. Estas tecnologías son:

- Algoritmos de búsqueda por fuerza bruta.
- Algoritmos de búsqueda heurísticos, entre ellos los algoritmos genéticos.
- Sistemas expertos.
- Lógica difusa.
- Redes neuronales artificiales.

Hay otros métodos dentro de la gran rama de la IA como las búsquedas estocásticas que son muy útiles en la identificación de patrones comunes en grandes grupos de muestras de datos, entre estas el método de Bayes, pero dichas tecnologías no fueron estudiadas por limitaciones de tiempo.

Para determinar cuál es la tecnología de IA que más se acoplaba a los requerimientos de este proyecto y que aportaba un nuevo enfoque a la forma en que el software IDS escogido procesaba los datos, se analizaron sin mayor detalle una por una.

Un estudio más minucioso hubiera requerido implementar todas las tecnologías existentes y por medio de pruebas de *benchmark* concluir cuál era la mejor, sin embargo, el alcance del proyecto no fue definido para tal efecto.

Los *algoritmos de búsqueda por fuerza bruta* y los *algoritmos de búsqueda heurísticos* fueron descartados, luego de analizar el código fuente de Snort [Arboleda & Bedón 2005] y descubrir que éste posee un motor de detección que utiliza búsqueda heurística de una forma bastante efectiva para encontrar la regla coincidente con el contenido de un paquete capturado, esto dentro del gran conjunto de reglas que Snort puede llegar a tener y de forma muy rápida.

Los *sistemas expertos* utilizan un motor de razonamiento y almacenan su conocimiento aparte en un conjunto de reglas. Snort se puede considerar un sistema experto basado en reglas, debido a que tiene un motor de detección que razona con un conocimiento almacenado en las reglas que contienen las firmas de ataque. Esta tecnología fue descartada, ya que no es muy enriquecedor adicionar un modulo que contiene un sistema experto a otro sistema experto.

La *lógica difusa* facilita el trabajo con datos o medidas inexactas y un sistema difuso puede tener características de aprendizaje automático. Igualmente las *Redes Neuronales Artificiales,* ofrecen la capacidad de abstraer y generalizar las características de un conjunto de datos de ejemplo (aprendizaje supervisado) o clasifican conjuntos de datos y sacan algunas características de éstos (aprendizaje no supervisado).

Tanto las RNA, como los sistemas difusos, ofrecen características que aportan un nuevo enfoque en el procesamiento de datos; sin embargo, 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.

2.4 ¿CÓMO INTEGRAR LA RNA A SNORT?

Snort ofrece tres formas de integrar módulos a su estructura para que funcionen en tiempo real o en línea (*on-line*), estas son:

- Elaborando un *preprocesador,* el cual da gran libertad a la hora de analizar y/o procesar los paquetes capturados. Además, se puede apoyar en otros preprocesadores disponibles por defecto con Snort para relalizar sus tareas.
- Elaborando un *plug-in de detección* para invocarlo a través de un archivo de reglas. Esto implica que el modulo sólo puede indicar si se cumple una condición o no, y esa respuesta es utilizada dependiendo de cómo esta formulada la regla.
- Con un *plug-in de salida* para formatear las salidas, pero este se limita a cambiar la forma en que la información es almacenada o mostrada al usuario.

Aparte de estas tres opciones para que el modulo funcione en tiempo real, se puede realizar un procesamiento fuera de línea (*off-line*), es decir, analizar el tráfico luego de que ha sido almacenado, así el proceso no tiene requerimientos rigurosos de tiempo. Existen proyectos que han utilizado redes neuronales para hallar patrones en tráfico capturado y almacenado (ver [Chavan et al. 2004]), el inconveniente es la imposibilidad de implementar un método de reacción inmediata ante un ataque de relativa larga duración, como un intento de Denegación de Servicio (DoS) por inundación de paquetes (*flood*) o un escaneo de puertos a muchos equipos.

La opción escogida fue la de elaborar un preprocesador, por la libertad que se tiene a la hora de diseñar el modulo. Un preprocesador permite acceder a los plugins de salida para generar alertas o almacenar el contenido de los paquetes capturados, también se puede utilizar la salida estándar o generar archivos, sin necesidad de apoyarse en los plugins de salida, para poner a disposición del administrador mas información sobre lo que pasa en la red.

2.5 ¿QUÉ TIPO DE ATAQUE DETECTAR CON EL MODULO INTELIGENTE?

Para la búsqueda del tipo de ataque específico, con el cual se probaría el modulo inteligente, se siguieron una serie de criterios. Estos son:

- Que el ataque, por las características que se puedan extraer de él; facilite la identificación, obtención y codificación de las entradas a la Red Neuronal Artificial.
- Que sea de ocurrencia frecuente.
- Preferiblemente, que se considere crítico debido al daño que pueda producir.
- Que la efectividad de Snort para detectar el ataque sea poca, es decir, que se presenten falsos positivos y/o falsos negativos.
- Y que el ataque permita hacer variaciones de modo que se pueda engañar a los métodos de detección convencionales de Snort.

Se pensó en detectar problemas que aquejan a los usuarios de Internet actualmente como el *Spam* y *Spyware*, los virus que se propagan por la red, los ataques de Denegación de Servicio (DoS), la explotación remota de software vulnerable, más específicamente, la explotación de desbordamientos de buffer (*Buffer Overflow* - BoF), o la búsqueda constante de equipos vulnerables por parte de algunos programas y/o personas maliciosas, es decir, escaneos de puertos.

Luego de un corto estudio se notó que problemas como el Spam y el Spyware (que están ligados entre si), y los virus, se escapaban del alcance del proyecto debido a su gran complejidad. Entonces se buscó detectar los ataques de Buffer Overflow, entre los cuales algunos de ellos ocasionan DoS.

Un ataque de desbordamiento de buffer (remoto) se caracteriza por la inyección de una cadena de Bytes a través de la red, entre los cuales va un conjunto de instrucciones

codificadas en ensamblador para que el equipo vulnerable, que recibe la cadena, las ejecute. Este conjunto de instrucciones en ensamblador es llamado *shellcode*, debido a que casi siempre busca ejecutar un *shell* (ventana de línea de comandos) con privilegios administrativos en el equipo remoto.

Se planteó entonces la detección de ataques de BoF (Buffer overFlow) a través de la identificación del *shellcode* que viaja por la red. Un *shellcode* tiene la apariencia que se ve a continuación:

```
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\xeb\x19\x5e\x31\xc9\x81\xe9\x89\xff"
```

En donde los Bytes con valor “\x90” (90 en base hexadecimal, es decir, 144 en base decimal) son utilizados para llenar el buffer vulnerable del equipo remoto y desbordarlo, de modo que las instrucciones dictadas por el atacante que están al final de la cadena (desde “\xeb” en adelante) queden por fuera del buffer donde serán ejecutadas.

Los valores que componen un shellcode son sumamente variables, y con herramientas como Metasploit (www.metasploit.com) se facilita aún más la elaboración y ocultación de éstos. Esto representa un problema a la hora entrenar una RNA que necesita identificar patrones comunes. También presenta problemas de rendimiento debido a que se necesitaría recorrer todo el *payload* (carga útil dentro de la trama de un protocolo, es decir, lo que queda luego de quitar sus encabezados) del protocolo en el cual se quiera identificar el BoF leyéndolo como una cadena (interpretando los distintos caracteres según un código que podría ser ASCII), por cada paquete que llegue al IDS. Por último se encontró gran dificultad para identificar en qué lugar se encontraba el

shellcode dentro del *payload* debido a que una instrucción que en hexadecimal es "\x41" puede significar INC %ECX en ensamblador (incrementar el registro C de un microprocesador x86) o puede ser la letra 'A' en ASCII.

Es así que se acogió el escaneo de puertos como el tipo de ataque específico, con el cual se probaría el modulo inteligente. El escaneo de puertos facilitó la identificación de las entradas de la RNA, es un ataque que ocurre muy frecuentemente, Snort no ha logrado reducir el número de falsos positivos generado y puede ser engañado con algunas de las numerosas variantes del ataque.

La gran mayoría de los ataques importantes a sistemas informáticos inicia por una etapa de reconocimiento, seguida por una etapa de identificación de vulnerabilidades y el escaneo de puertos es una herramienta fundamental en estas dos etapas, convirtiéndose por regla general en el aviso de un, muy probable, ataque futuro. De esta forma el sistema pretende detectar un ataque desde su gestación, ofreciendo una solución preventiva más que correctiva.

3 MODULO INTELIGENTE PARA LA DETECCIÓN DE ESCANEOS DE PUERTOS - PortscanAI

En este capitulo se describe el análisis realizado al problema del escaneo de puertos para identificar las características que permitieron detectarlo con el preprocesador PortscanAI. También son tratados el diseño y algunos aspectos de la implementación del módulo.

3.1 ANÁLISIS DEL ESCANEO DE PUERTOS

La primera consideración a tomar para la detección de un escaneo de puertos es que se debe hacer un análisis *stateful*, es decir, se 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. Es aquí donde el concepto de *flow* o flujo resulta de mucha utilidad.

3.1.1 ¿Qué es un *flow?*

Un *flow* de IPv4 es único cuando el protocolo que va sobre IP (ICMP, TCP o UDP), la dirección IP de origen, el puerto TCP de origen, la dirección IP de destino y el puerto TCP de destino son los mismos, sin importar el orden de origen y destino. Un *flow* identifica un conjunto de paquetes que se intercambian entre dos equipos a través de unos puertos dados.

Por ejemplo, si se tiene dos equipos A y B, cada uno con dirección IP distinta, y se envía un paquete de A por el puerto 20 (nótese A(20)) hacia B, que lo recibe en el puerto 30 (nótese B(30)); ese paquete pertenecerá a un *flow* que se puede identificar

como flow1. Si B responde a A utilizando los mismos puertos, es decir de B(30) hacia A(20), dicho paquete de respuesta pertenecería al mismo flow1. Pero si se envía un paquete de A(20) hacia B(31) entonces se crea un nuevo *flow* que es el flow2. Si apareciera un computador C con distinta dirección IP y se enviara un paquete de A(20) a C(30) ese sería un flow3.

En el caso del protocolo TCP, que es orientado a la conexión, la creación de un nuevo *flow* se da siempre que se inicia una nueva conexión, en otras palabras, se puede detectar el número de inicios de conexión o nuevos intentos de comunicación (para el caso UDP) detectando los nuevos *flows* creados.

3.1.2 Prototipo utilizado para el análisis

Para el análisis del tráfico de la Universidad del Cauca en busca de la caracterización de los escaneos de puertos, se realizó un prototipo que facilitó la obtención y observación de distintas variables de medición. El prototipo implementado como un preprocesador de Snort sirvió a su vez como base para el módulo PortscanAI.

En la **Figura 3.1** se puede ver el prototipo de análisis de una forma general. Su funcionamiento es como sigue: cuando un paquete es capturado por Snort, el primer filtro al que es sometido es el Decodificador, el cual, descarta los paquetes con encabezados erroneos y notifica de su ocurrencia a través de alertas. Los paquetes que pasan son tomados por los preprocesadores, entre los cuales se encuentra el prototipo de análisis.

El preprocesador prototipo se apoya en otro preprocesador que viene por defecto con Snort, llamado Flow. El Flow, como su nombre lo dice, se encarga de manejar los *flows* que se identifican en los paquetes capturados. En este caso, Flow se encarga de llamar un método en el preprocesador prototipo cada vez que llega un paquete que pertenece a un nuevo *flow* y le entrega a dicho preprocesador el primer paquete de ese nuevo

flow. En otras palabras, el prototipo recibe sólo los paquetes que intentan realizar una nueva conexión.

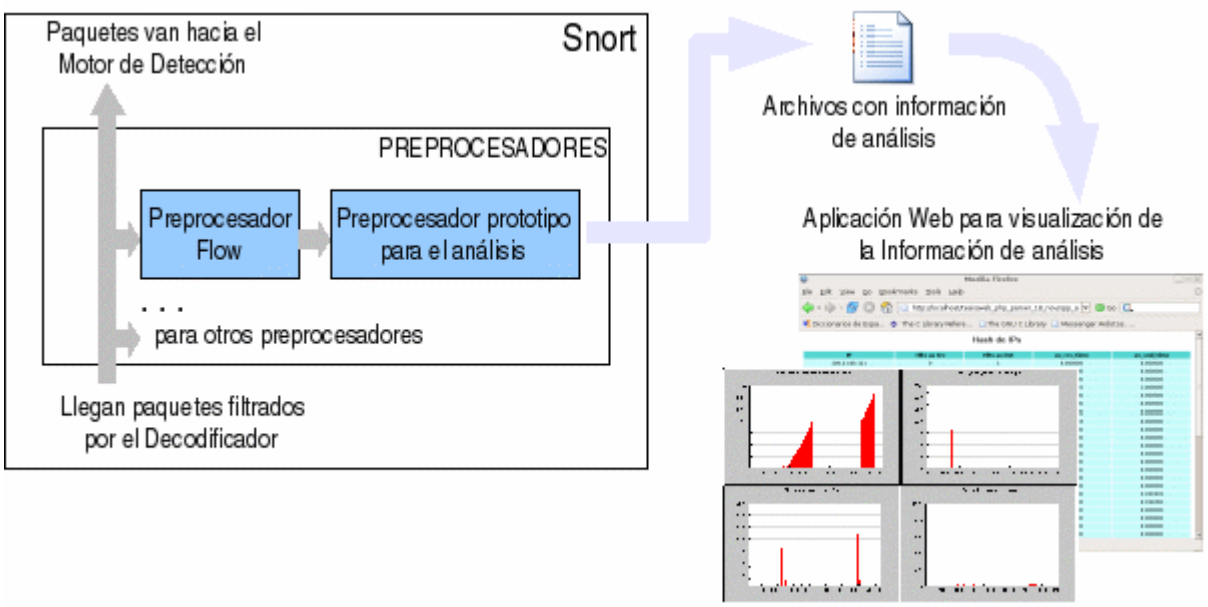


Figura 3.1. Diagrama modular general del prototipo desarrollado para el análisis del tráfico

De todos estos paquetes recibidos se sacan mediciones por cada dirección IP observada, como el número de intentos de conexión (nuevos *flows*) que ha recibido una dirección IP. También se sacan medidas para dos direcciones IP relacionadas, como el número de intentos de conexión enviados de una IP hacia otra. Estas medidas son guardadas en archivos cuyos datos son analizados luego por medio de una aplicación Web desarrollada en PHP.

En la **Figura 3.2** se puede ver el funcionamiento interno del preprocesador desarrollado. Las mediciones obtenidas son guardadas en dos tablas Hash: una utiliza como clave una dirección IP (Tabla Hash de IPs), de modo que cada ves que llega un paquete se crean o actualizan dos registros o filas de la tabla, correspondientes a la dirección de origen y la dirección de destino. La otra tabla utiliza como clave el par IP origen e IP destino (Tabla Hash de relaciones), entonces se crea o actualiza un registro

de la tabla cada vez que llega un paquete para llevar la cuenta de cuantos intentos de conexión se ha enviado de una IP a otra y así ver la relación entre estas.

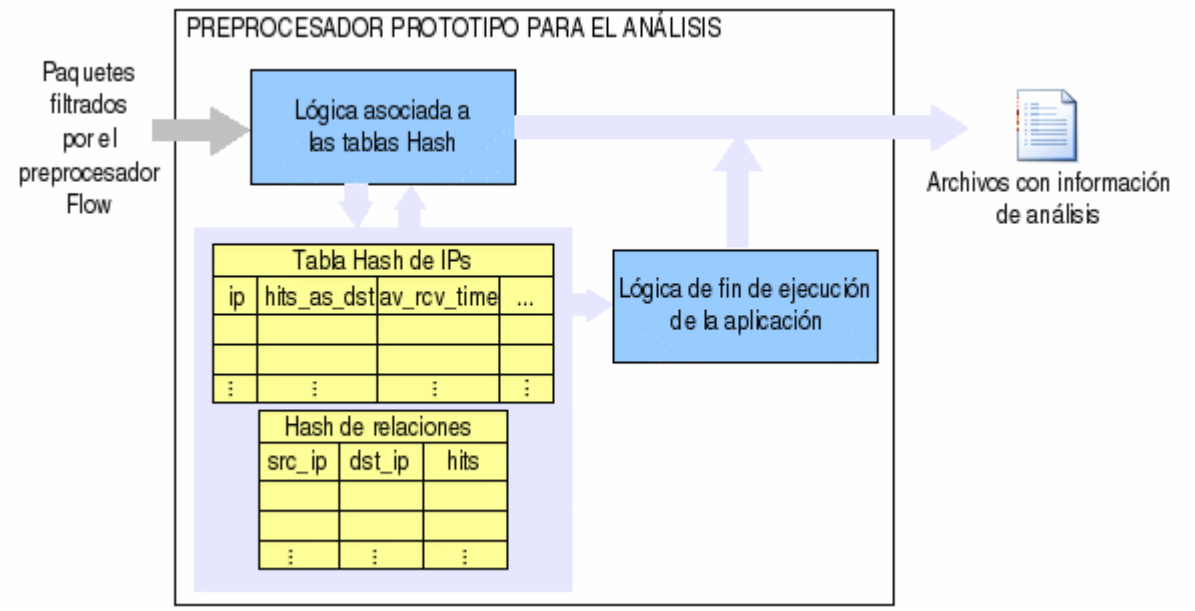


Figura 3.2. Diagrama modular interno del prototipo desarrollado para el análisis del tráfico

La creación y actualización de los registros de las tablas Hash está a cargo del módulo de Lógica asociada a las tablas Hash. Este escribe en un archivo, cada vez que arriba un paquete, datos del estado actual de las tablas, útiles para generar gráficas que muestran la evolución de las medidas que se encuentran en ellas. Al salir del programa, la Lógica de fin de ejecución de la aplicación toma las medidas finales consignadas en las tablas Hash y las escribe en un archivo aparte para mostrar la lista de IPs analizadas y las relaciones entre estas.

3.1.3 Mediciones y resultados obtenidos del análisis

Para realizar las mediciones 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.

Dentro del grupo anteriormente mencionado se debe pesar la cantidad (cuántos son enviados) y la temporalidad (cada cuanto se efectúa esta operación). Entonces se decidió obtener los siguientes datos por cada dirección IP capturada:

- **hits_as_dst** (*hits as destination* – *hits* como destino): número de paquetes de inicio de conexión (*flows*) 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.

- **hits_as_src** (*hits as source* – *hits* como origen): número de paquetes de inicio de conexión (*flows*) 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.

- **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 escanedo.

- **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.

- **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.

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

- **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.

En la **Figura 3.3** se pueden observar los parámetros capturados en las tablas Hash para diferentes direcciones IP. Las columnas **rst_resp** y **win_count** del Hash de IPs no son utilizadas en las versiones entregadas del prototipo de análisis, ni en PortscanAI. Estas significan: el número de paquetes con la bandera RST enviados por la dirección IP y el número de ventanas que han transcurrido (cuantas veces se han reiniciado las mediciones), respectivamente.

Hash de IPs

IP	hits_as_src	hits_as_dst	av_rcv_time	av_snd_time	negative_resp	rst_resp	win_count
10.1.13.18	1	0	0	0	0	0	0
10.1.13.19	1	0	0	0	0	0	0
10.1.13.20	1	0	0	0	0	0	0
10.1.13.21	1	0	0	0	0	0	0
10.1.13.22	1	0	0	0	0	0	0
10.1.13.221	5	0	0	12.0497	0	0	0
10.1.13.38	0	64	0.000685	0	121	0	1
10.200.1.130	0	72	0.00023	0	109	0	1
10.200.2.112	0	1	0	0	0	0	0
10.200.2.114	2	0	0	0.0005	0	0	0
10.200.2.119	1	0	0	0	0	0	0
10.200.2.136	4	0	0	9.15786	0	0	0

Hash de Relaciones Directas

IP src	IP dst	Número de Hits
10.1.13.38	172.16.130.87	1
10.200.1.130	172.16.130.195	1
10.200.1.130	172.16.130.87	1
10.200.2.112	172.16.42.41	1
10.200.2.173	172.16.130.127	1
10.200.2.173	172.16.130.169	1
10.200.2.174	172.16.130.127	1
10.200.2.174	172.16.130.169	6
10.200.2.174	172.16.130.183	1
10.200.2.174	172.16.130.195	1
10.200.2.32	194.83.71.244	1
10.200.2.32	200.115.114.30	1
10.200.2.32	201.34.232.26	1
10.200.2.32	202.74.220.123	1
10.200.2.32	208.195.210.141	43

Figura 3.3. Tablas con los parámetros capturados.

En la **Figura 3.4** se encuentran las gráficas que muestran la evolución de dichos parámetros para una dirección IP dada.

En la gráfica superior izquierda (hits_as_dst vs. No de *flows* capturados en total), se identifica en el eje X el número total de inicios de conexión detectados por el sistema y en el eje Y, el número de inicios de conexión que ha recibido el host cuya dirección es la correspondiente a este conjunto de gráficas. En la inferior izquierda (rcv_time vs. No de *flows* capturados en total) se puede ver en el eje Y (rcv_time), el tiempo transcurrido entre llegadas consecutivas de inicios de conexión (*flows*) al host con esta dirección IP.

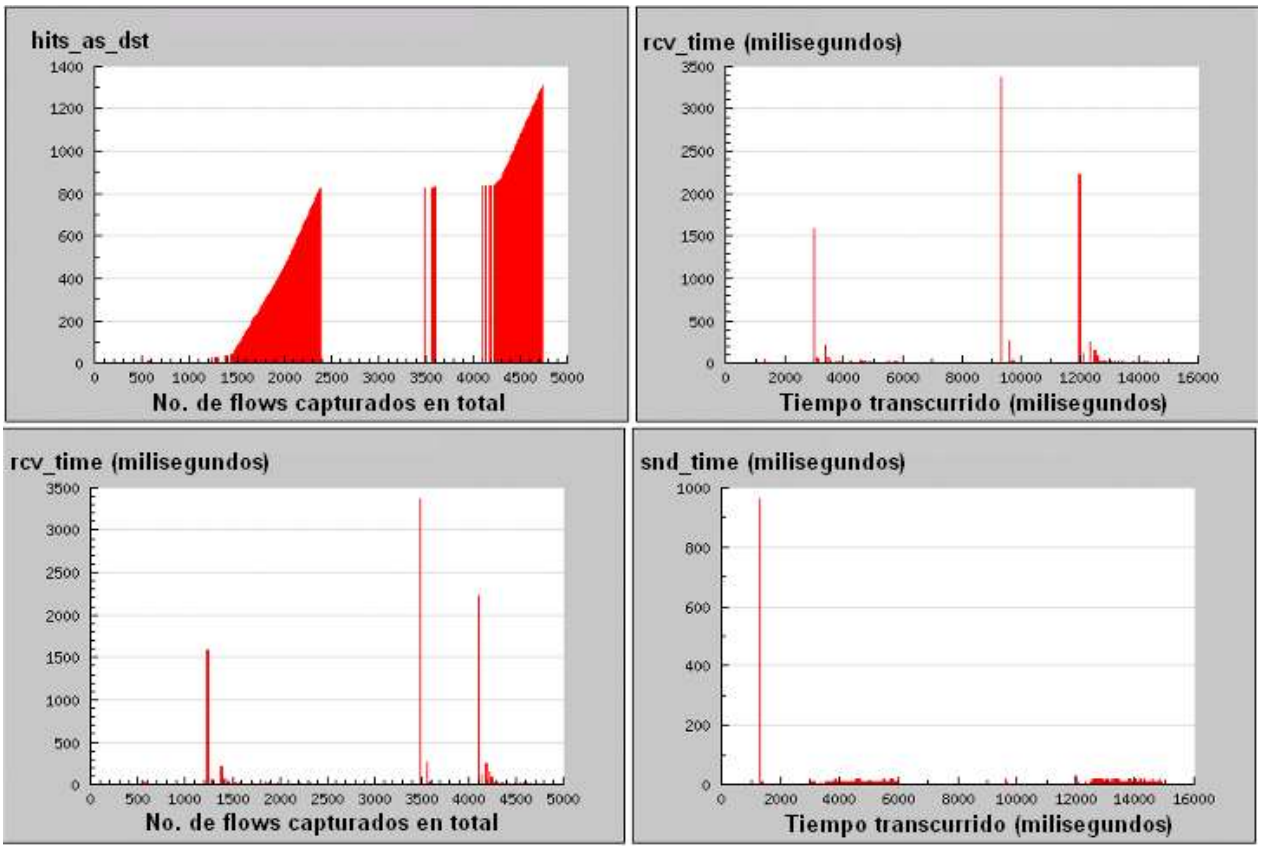


Figura 3.4. Gráficas de parámetros para una dirección IP dada.

Al ver las dos gráficas mencionadas se puede observar en la primera de ellas (hits_as_dst vs. No de *flows* capturados en total), que hubo un incremento en el número de inicios de conexión recibidos por este PC entre el *flow* capturado número 1500 y el 2500. Luego cesaron los inicios de conexión recibidos. Al mismo tiempo en la segunda gráfica (rcv_time vs. No de *flows* capturados en total) se puede apreciar que entre el *flow* número 1500 y el 2500, los tiempos entre los inicios de conexión recibidos fueron mínimos, es decir, que llegaron de una forma muy rápida y seguida al PC con la dirección IP estudiada.

3.1.4 Problemas con el uso de promedios

Entre los parámetros definidos para hacer las mediciones hay dos que son el resultado de promediar los datos capturados. Estos son **av_rcv_time** (*average receive time* – tiempo promedio de recepción) y **av_snd_time** (*average send time* – tiempo promedio de envío).

Existe un problema con la utilización de promedios en grandes conjuntos de datos. Cuando el conjunto crece demasiado el resultado del promedio tiende a aproximarse a los valores que más se repiten dejando de lado los valores que representan una minoría dentro del conjunto. En el caso de los parámetros en cuestión, son esos valores minoritarios los que importan a la hora de estudiar las numerosas muestras capturadas durante un largo periodo de funcionamiento, debido a que las muestras generadas por un ataque son muy pocas en comparación con las numerosas muestras generadas por el tráfico normal.

La solución a este problema fue reducir los conjuntos sobe los cuales se calcula el promedio, a través de la utilización de una ventana, de modo que representara con mayor fidelidad el comportamiento de los datos.

Inicialmente se planteó que la ventana fuera de tiempo, es decir, capturar datos durante el periodo de tiempo que durara la ventana, calcular el promedio y reiniciar las medidas; pero luego de notar que una ventana de tiempo provoca una variabilidad muy alta en el tamaño de los conjuntos de muestras (conjuntos pequeños cuando no hay tráfico y conjuntos gigantescos cuando hay mucho tráfico), se optó por medir la ventana, ya no en tiempo, sino en *hits* como destino (hits_as_dst) para cada dirección IP. Medir la ventana de esta forma garantiza un tamaño aproximadamente uniforme de los conjuntos de muestras, así la ventana dura poco cuando hay abundante tráfico y se demora cuando el tráfico es escaso.

3.2 DISEÑO DE PortscanAI

El preprocesador PortscanAI permite detectar escaneos de puertos, adicionalmente puede arrojar estadísticas y gráficas, como característica opcional, para el análisis del tráfico de red por parte del administrador. Está implementado en lenguaje ANSI C siguiendo los lineamientos de programación de la comunidad de desarrolladores de Snort.

En la **Figura 3.5** se muestra el funcionamiento general del sistema implementado, este tomó como base el prototipo para el análisis del tráfico y es por eso su parecido. El preprocesador PortscanAI toma los nuevos *flows*, busca si hay indicios de escaneo de puertos y de detectar alguno, lo reporta como una alerta en el archivo de alertas generado por Snort. Opcionalmente genera información de análisis en archivos y en una Base de Datos (BD) para que sea revisada por el administrador de red con ayuda de la aplicación Web destinada para ello.

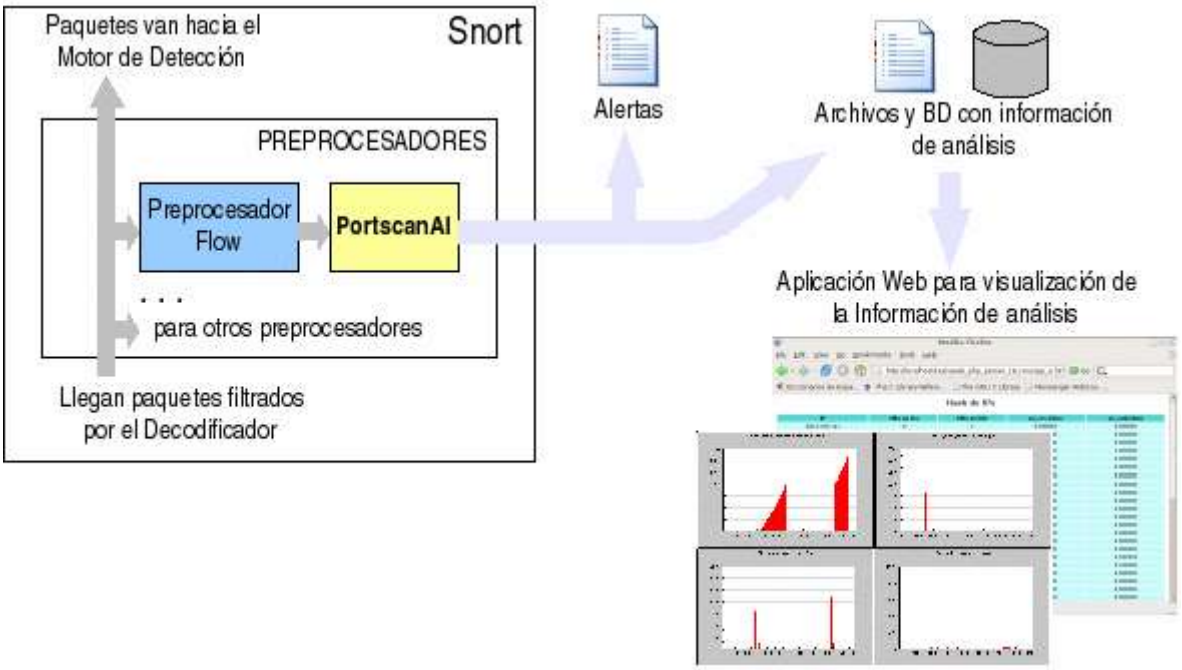


Figura 3.5. Diagrama general de PortscanAI.

En la **Figura 3.6** se representa el funcionamiento interno de PortscanAI. Cuando llega un nuevo *flow*, el módulo de Lógica asociada a las tablas Hash saca los parámetros expuestos en la sección de Mediciones y resultados obtenidos del análisis, guardándolos en las tablas Hash de IPs y de relaciones. Dicho módulo también crea opcionalmente un conjunto de archivos que sirven para generar gráficas (iguales a las generadas por el prototipo de análisis).

El módulo de Adecuación de los datos para la RNA toma los parámetros obtenidos por el módulo anterior, los normaliza y los relaciona para obtener las entradas que irán a la RNA (ver la sección **3.2.1 Elección de las entradas de la red neuronal**), la cual, según como haya sido entrenada, dará un juicio representado en dos números entre 0 y 1: el valor de tráfico normal y el valor de escaneo de puertos. Por ejemplo, la red podría indicar con dichas salidas que está un 20% (0.2) segura que el tráfico es normal y un 80% (0.8) segura que hay un escaneo de puertos.

La sección de Filtrado toma la salida de tráfico normal y si ésta tiene un valor menor al de un umbral o nivel de sensibilidad configurable por el administrador, entonces se genera la alerta indicando que se está realizando un escaneo de puertos.

La Lógica de fin de ejecución de la aplicación realiza lo mismo que en el prototipo de análisis, sólo que esta ves no genera un archivo sino una Base de Datos, lo cual da mas libertad a la hora de analizar con la aplicación Web.

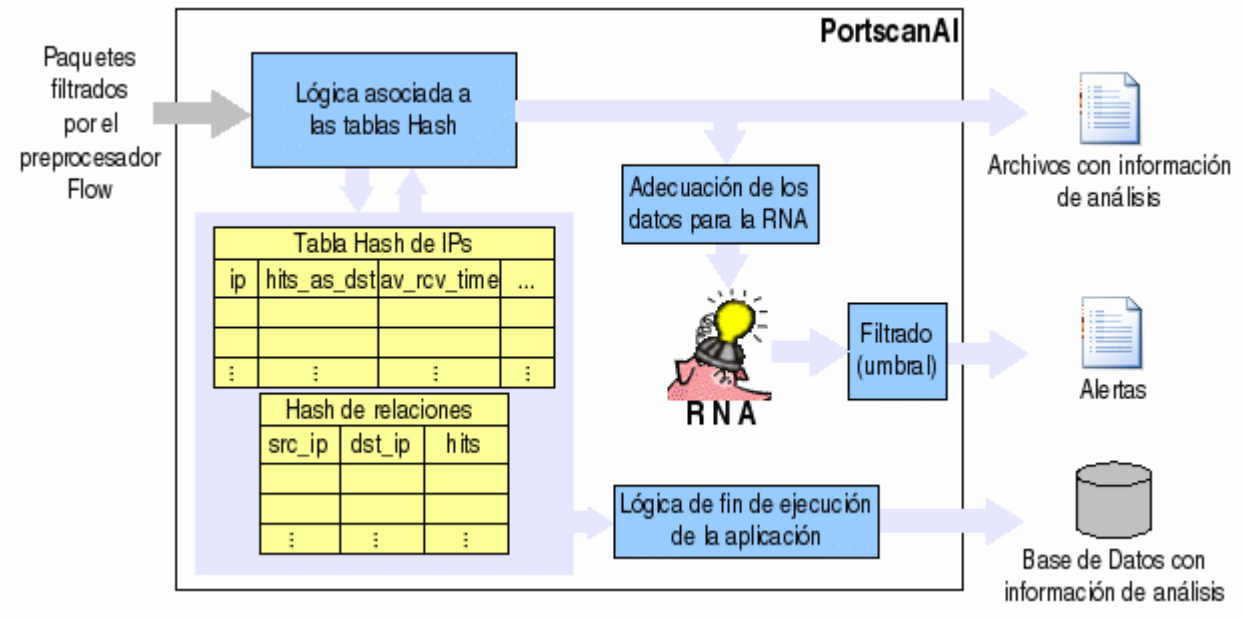


Figura 3.6. Diagrama detallado de PortscanAI.

3.2.1 Elección de las entradas de la red neuronal

La elección de las entradas de una red neuronal es el paso inicial a la hora de trabajar con este tipo de tecnología, de esto depende si la red obtendrá la información necesaria para tener un entrenamiento y ejecución óptimos.

Se ensayaron diferentes combinaciones de entradas y se observó el error producido por estas durante el entrenamiento. Inicialmente se probaron entradas no normalizadas y se ingresaron los parámetros de la sección de Mediciones y resultados obtenidos del análisis, sin lograr obtener una reducción adecuada del error. Luego de esto se optó por normalizar las entradas y sacar relaciones entre los parámetros obtenidos en lugar de introducirlos a la red directamente, siendo esta la opción que mejor se comportó.

Finalmente, se seleccionaron las siguientes entradas para la red:

Para la descripción de las entradas se asumirá que se ha capturado un inicio de conexión (*flow*) que va del origen A hacia el destino B.

No.	***Entrada***	***Descripción***
0	hits_as_dst/max_hits_as_dst	Hits como destino (*hits_as_dst*) normalizado. *max_hits_as_dst* es el máximo valor de este parámetro en toda la tabla Hash de IPs. Indica que tan significativo es el número de inicios de conexión recibidos por B en comparación con el destino que más ha recibido inicios de conexión durante el funcionamiento del sistema.
1	hits_as_src/max_hits_as_src	Hits como origen (*hits_as_src*) normalizado. *max_hits_as_src* es el máximo valor de este parámetro en toda la tabla Hash de IPs. Indica que tan significativo es el número de inicios de conexión enviados por A en comparación con el origen que más ha enviado inicios de conexión durante el funcionamiento del sistema.
2	av_rcv_time	Tiempo promedio de recepción. Indica el tiempo promedio entre inicios de conexión recibidos consecutivamente por B.
3	av_snd_time	Tiempo promedio de envío. Indica el tiempo promedio entre inicios de conexión enviados consecutivamente por A.
4	negative_resp/max_negative_resp	Número de respuestas negativas (*negative_resp* o *ack_rst_resp*) normalizado. Indica que tan significativo es el número de respuestas negativas enviadas por A en comparación con el origen que más ha enviado respuestas negativas durante el funcionamiento del sistema
5	rel_hits/hits_as_src	Hits de relación (*rel_hits*) sobre hits como origen. Indica qué porcentaje del total de los inicios de conexión enviados por A han ido hacia B.
6	rel_hits/hits_as_dst	Hits de relación sobre hits como origen. Indica qué porcentaje del total de los inicios de conexión recibidos por B han venido de A.

Tabla 3.1. Entradas de la Red Neuronal Artificial.

3.2.2 Generación de los sets de entrenamiento

La generación de los sets de entrenamiento es una parte esencial en la elaboración de cualquier sistema que involucre una Red Neuronal Artificial. Un set de entrenamiento consiste en un conjunto de pares de datos que corresponden a las entradas y a las salidas de la RNA (Red Neuronal Artificial).

Los sets de entrenamiento fueron generados utilizando el propio Snort por medio de la adaptación de uno de los prototipos realizados anteriormente implementado como un preprocesador. Las condiciones en las que se efectuó este proceso fueron controladas, de modo que las salidas de la red en los sets de entrenamiento correspondieran a valores reales de tráfico.

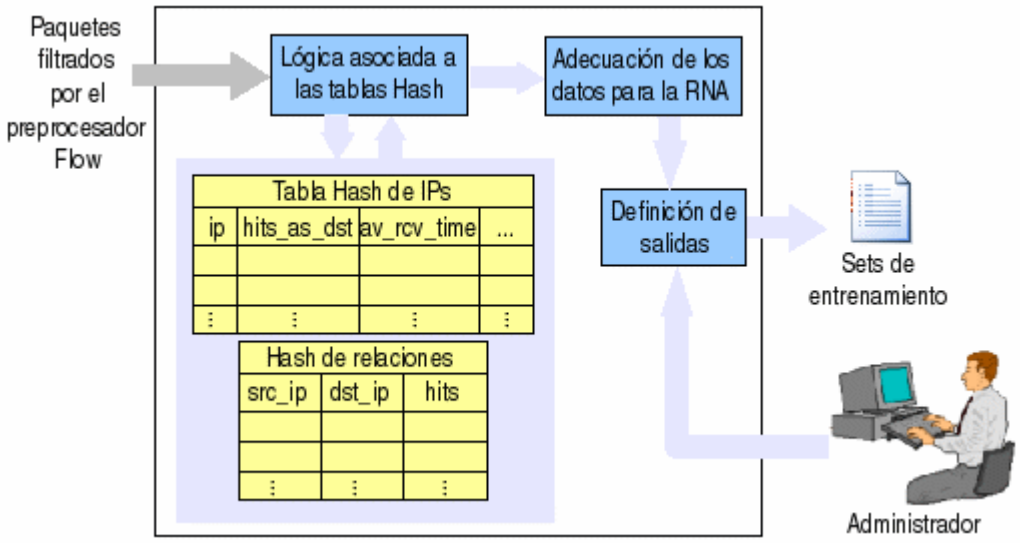


Figura 3.7. Preprocesador para la generación de sets de entrenamiento

En primer lugar se dejó a Snort olfateando en horario de oficina asegurándose que no hubiera escaneos de puertos. Los patrones de entrada generados durante ese periodo se asociaron a la salida de la RNA que indica tráfico normal. El segundo paso fue iniciar ataques con diferentes técnicas para el escaneo de puertos, los patrones generados fueron asociados con salidas que indicaban el estado de ataque.

Al generar los patrones de esta manera, el sistema identifica de forma autónoma las características reales del tráfico normal y del tráfico que contiene escaneos de puertos, asegurando adaptabilidad para los distintos comportamientos que se encuentren en una red. Todo esto gracias a la capacidad de aprendizaje que posee la Red Neuronal.

Los patrones obtenidos fueron guardados en un archivo de forma adecuada para que el software de simulación de redes neuronales SNNS (Stuttgart Neural Network Similator) o JavaNNS pudiese interpretarlos. Las líneas del archivo tienen el siguiente formato.

Para tráfico normal:

```
#In: h_dst  h_src   a_r_time  a_s_time   n_resp    rh/has    rh/had
0.470588  0.187500  1.930544  5.086774  1.000000  1.000000  0.375000
#Out: Normal ataque
1 0
```

Para un escaneo:

```
#In: h_dst  h_src   a_r_time  a_s_time   n_resp    rh/has    rh/had
1.000000  1.000000  0.001508  0.033645  1.000000  0.996689  0.537500
#Out: Normal ataque
0 1
```

En un primer intento se generaron 4449 patrones, de los cuales 2740 pertenecían a tráfico normal y 1709 a escaneo.

3.2.3 Elaboración de la red neuronal

Para la simulación de la RNA se utilizó el software JavaNNS 1.1 que es una versión en Java basada en SNNS 4.2 (Stuttgart Neural Network Simulator). Este software permite

modelar diversos tipos de redes neuronales de una forma gráfica, entrenarlas, simular su comportamiento y eventualmente exportarlas a código C.

Se probó con diversos diseños de RNA, todos utilizando un archivo de set de datos de entrenamiento y uno de validación.

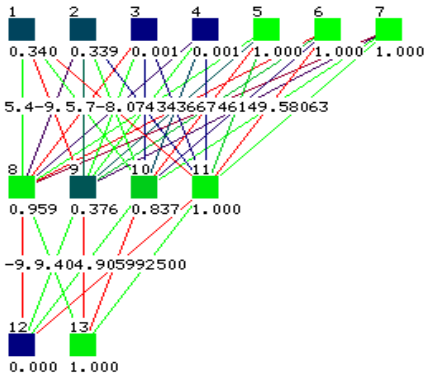


Figura 3.8. Red MLP de 7 entradas y dos salidas.

Inicialmente se probaron diseños de redes cuyas entradas no eran normalizadas y con una sola salida, estas redes no convergieron (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 normalizada por cada uno de los estados que arrojara la red. Todas las capas de las redes utilizaban la **función de activación sigmóide** (llamada ***Act_logistic*** en SNNS).

Con una red MLP (Multi-Layer Perceptron) de siete entradas, cinco de ellas normalizadas, y dos salidas también normalizadas (0 1 para ataque y 1 0 para tráfico normal) como la que se ve en la **Figura 3.8** se obtuvo una gráfica de error como la que se ve en la **Figura 3.9**. La **función de aprendizaje** fue ***Backpropagation*** o función de retropropagación del error.

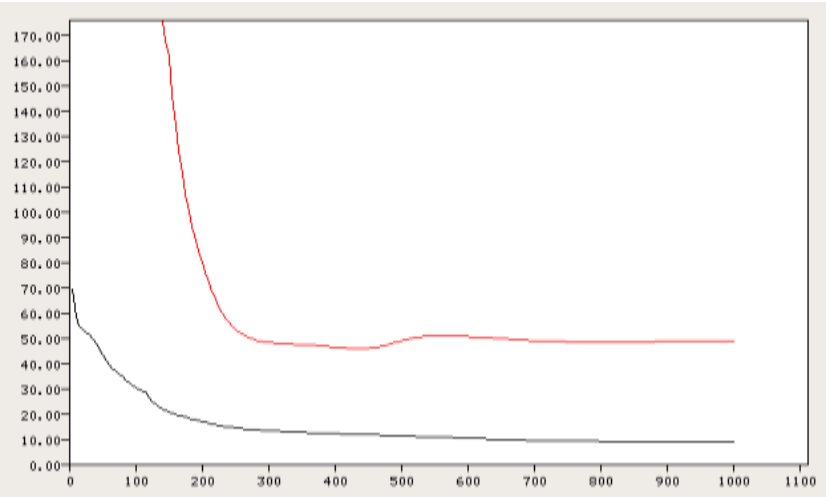


Figura 3.9. Error de entrenamiento (curva negra) y validación (curva roja) en la red MLP.

Se puede ver que el error producido por los sets de entrenamiento (curva de color negro) siempre desciende, indicando que la red está aprendiendo cada uno de los patrones especificados en los sets de entrenamiento. Mientras que el error de los sets de validación primero desciende y luego se incrementa, indicando que la red se sobreentrena, es decir, pierde su capacidad de generalización luego de los 450 ciclos de entrenamiento.

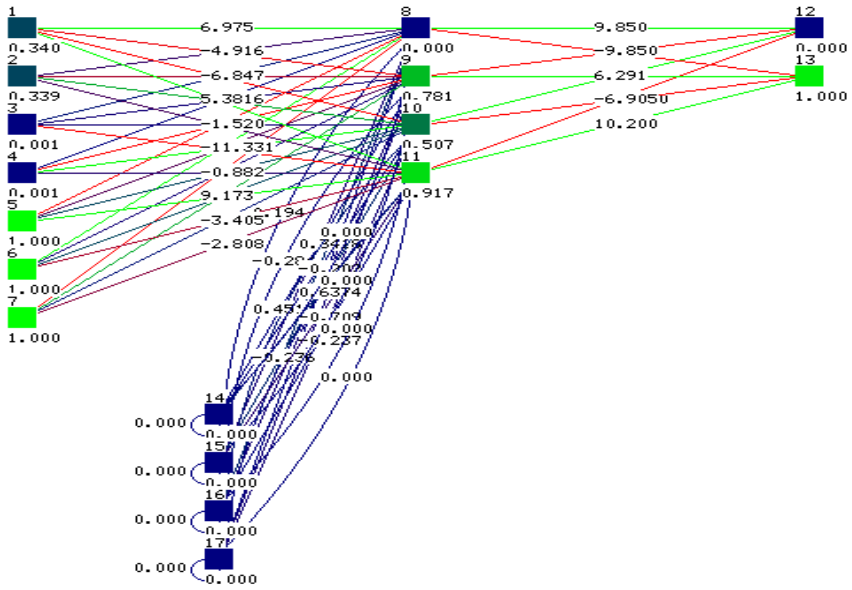


Figura 3.10. Red Elman con una capa de contexto compuesta por cuatro neuronas en parte inferior.

En una red Elman con un contexto asociado a su capa oculta, entradas y salidas iguales a la anterior, como la que se ve en la **Figura 3.10**, se obtuvo la gráfica de error que se muestra en la **Figura 3.11**. Se utilizó el algoritmo de SNNS (JavaNNS no permite crear redes Elman) llamado ***JE_BP*** **(Jordan and Elman BackPropagation)** o algoritmo de retropropagación del error adaptado para redes parcialmente recurrentes, como son las redes Jordan y Elman.

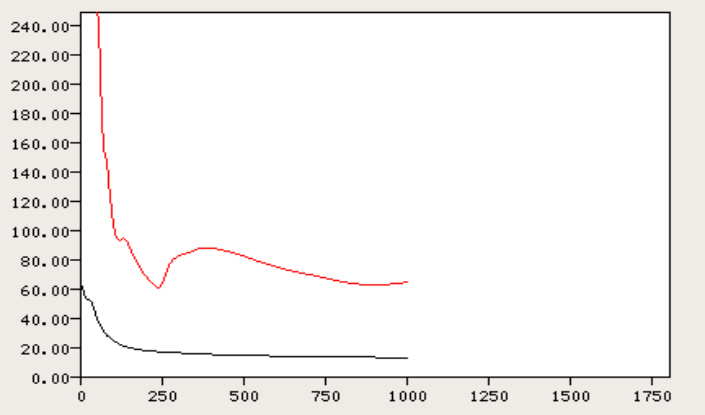


Figura 3.11. Error de entrenamiento y validación en la red Elman.

En la red Elman se obtuvo un aprendizaje óptimo a las 250 iteraciones, como lo muestra el mínimo en la curva de error de validación (color rojo). Esto no representa una mejoría significativa con respecto a la red MLP.

3.3 ASPECTOS DE IMPLEMENTACIÓN

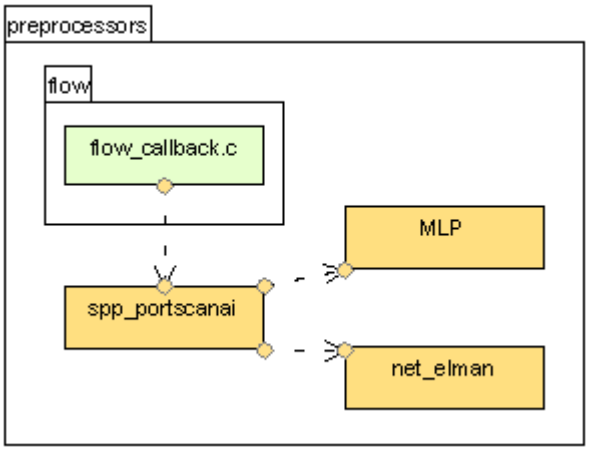


Figura 3.12. Dependencia entre los archivos fuente.

La implementación en lenguaje C de PortscanAI sigue los lineamientos definidos por la comunidad de Snort para adicionar nuevos preprocesadores al IDS [Bedón & Arboleda 2005]. La totalidad del código del preprocesador, excepto la parte de la RNA, está en el archivo “preprocessors/spp_portscanai.c” que va acompañado de su archivo de cabecera “preprocessors/spp_portscanai.h” utilizado para declarar estructuras de datos y exportar algunas funciones, de modo que sean accesibles desde otros archivos.

El código de las dos redes neuronales (MLP y Elman) se encuentra en los archivos “preprocessors/MLP_x.c” y “preprocessors/net_elman_x.c” acompañados de sus cabeceras. Estos archivos fueron generados con la herramienta snns2c a partir de los archivos .net en los cuales SNNS (simulador de redes neuronales) guarda las redes

neuronales ya entrenadas. La herramienta snns2c viene con la distribución de SNNS v4.2.

Se tuvo que modificar el archivo "preprocessors/flow/flow_callback.c" del preprocesador Flow con el fin de acoplar PortscanAI. En la **Figura 3.12** se puede ver la dependencia entre los archivos de código fuente, en donde flow_callback.c depende de spp_portscanai y así sucesivamente.

3.4 PROCESO DE DESARROLLO SOFTWARE DEL MÓDULO

Para el desarrollo del módulo se siguió un proceso de desarrollo personalizado, se estudiaron procesos de desarrollo como RUP (*Rational Unified Process* – Proceso Unificado de Rational) y XP (*eXtreme Programming* – Programación Extrema) pero fueron descartados ya que la orientación de la programación es estructurada y el equipo de desarrollo estaba compuesto por sólo dos personas.

En el desarrollo del módulo se siguió un proceso iterativo e incremental. Se realizó un prototipo al cual se le iba aumentando la funcionalidad y corrigiendo los errores. Se realizaba una copia de seguridad luego de cada sesión de trabajo la cual era etiquetada con la fecha y un indicador que decía si la versión era estable o no.

El prototipo fue utilizado en tres etapas, la primera versión funcional fue utilizada para analizar el tráfico de la red con el fin de entender los escaneos de puertos (ver **3.1 ANÁLISIS DEL ESCANEO DE PUERTOS**), luego se adicionó la habilidad de generar sets o conjuntos de patrones de entrenamiento para la red neuronal. En esta etapa se hacían pruebas con las distintas topologías de redes neuronales con el software SNNS, utilizando los sets de entrenamiento generado por el prototipo (ver **3.2.2 Generación de los sets de entrenamiento**). Finalmente, cuando ya se tenían redes neuronales entrenadas y estables, dichas redes fueron transformadas a código C y adicionadas al prototipo formandose la versión final de PortscanAI (ver **3.2 DISEÑO DE PortscanAI**).

De forma paralela al desarrollo del preprocesador prototipo de Snort, se iba desarrollando la consola de análisis de estadísticas, que es una aplicación Web en PHP para analizar los logs generados por el preprocesador. Esta fue útil para la depuración del módulo y es útil para el análisis del tráfico de la red por parte del administrador.

Se encontraron problemas en el desarrollo por falta de conocimientos en lenguaje C, pero gracias al análisis del código fuente de Snort se resolvieron muchas dudas a partir de los algoritmos que componen dicho software, los cuales sirvieron como ejemplo.

4 PLAN DE PRUEBAS Y RESULTADOS

Para determinar la calidad de la solución realizada durante el proyecto se hicieron pruebas de rendimiento y efectividad de detección de la aplicación. Estas pruebas se llevaron a cabo en una de las salas de cómputo del Departamento de Sistemas de la Universidad del Cauca. Todos los equipos utilizados (sensores, víctimas y atacantes) se encontraban en el mismo segmento de red (conectados al mismo Hub).

Características del host en el que se corrió Snort para realizar las pruebas (sensor):

Microprocesador	Intel Pentium 4 de 1.5 GHz
RAM	630 MB
Velocidad de conexión ethernet	10 Mbps
Distribución de Linux	Debian Sarge
Versión del kernel	2.4.27

La configuración del preprocesador **sfPortscan** para todas las pruebas fue:

Detectar en todos los protocolos: `proto { all }`
Reportar todos los tipos de escaneo
Nivel de sensibilidad bajo: `sense_level { low }`

La configuración del preprocesador **PortscanAI** para todas las pruebas fue:

Ignorar tráfico de difusión: `ignorebc 1`
Límite bajo ventana 100, alto 1600: `analyze_thr_lower 100 analyze_thr_upper 1600`
Nivel de sensibilidad 0.05 `sense_level 0.05`

4.1 PRUEBAS DE RENDIMIENTO

4.1.1 Porcentaje de paquetes perdidos

Es el porcentaje de paquetes sin analizar o *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.

Condiciones:

Número de paquetes capturados:	109 000
Tiempo que duró la captura:	60 seg.

Resultados:

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

4.1.2 Consumo de memoria

- **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.
- **Con tráfico:** incremento en el consumo de memoria luego de someter el sensor a tráfico generado de forma controlada.

Condiciones:

Tipo de tráfico utilizado:	Se enviaron 1676 paquetes TCP desde un sólo origen, cada uno dirigido a un puerto distinto del sensor.

Resultados:

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

Tabla 4.1. Consumo de memoria.

4.2 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.

4.2.1 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.

Ninguna de las alertas generadas tanto por sfPortscan como por la red MLP y la red Elman coincidieron, cada sistema de detección arrojó alertas distintas.

- **Escaneos detectados:** cuántos escaneos detectaron los sensores.
- **Falsas alarmas:** indica cuántos de los escaneos reportados anteriormente fueron falsas alarmas confirmadas (falsos positivos). Los ataques que no son considerados falsas alarmas no fueron confirmados, lo cual quiere decir que pueden ser aún falsas alarmas.
- **%de paquetes perdidos:** porcentaje de paquetes *dropped* por el sensor con la red MLP y el sensor con la red Elman.

Condiciones:

Fecha de la prueba:	Miércoles 25 de enero de 2006
Hora de inicio:	4:15 PM
Hora de fin:	5:34 PM
Tiempo que duró la captura:	1 h – 18 m – 40 s
Número de paquetes capturados:	2’585 956

Resultados:

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

Tabla 4.2. Efectividad de la detección con tráfico normal.

4.2.2 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).

1. **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>

2. **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>

3. **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.

   nmap –sS –P0 –D <conjunto de IPs señuelo> <IP victima>

4. **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.
   nmap –sT –P0 <IP victima>

5. **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.

   nmap –sT –P0 <rango de IPs victima>

Resultados:

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

Tabla 4.3. Efectividad de la detección bajo escaneo.

5 RECOMENDACIONES PARA LA IMPLANTACIÓN DE UN NIDS EN LA UNIVERSIDAD DEL CAUCA

En este capitulo se proponen algunas recomendaciones para la implantación de un NIDS, más concretamente Snort, en la red de datos de la Universidad del Cauca, basado en un estudio parcial del entorno de operación y de los requerimientos de seguridad que presenta la red de la universidad, de acuerdo con lo expuesto por [Herrera 2003] y [Cox & Gerg 2004]. No se pretende que sea un plan completo para la implantación de un sistema de detección de intrusos, ya que esto implicaría la comparación de distintas opciones de productos NIDS y de diversos proveedores por medio de un estricto plan de pruebas, además se necesitaría un estudio profundo del tráfico, del personal de administración y de los puntos mas sensibles de la red, a los cuales no se tiene acceso por cuestiones de seguridad.

5.1 ¿Para qué un NIDS en la Universidad del Cauca?

La implantación de un sistema de detección de intrusos en la Universidad del Cauca podría ser muy útil, si se le usa y configura adecuadamente. Sería un control de seguridad que serviría de apoyo para los equipos encargados de la respuesta a incidentes. Podría dar más información sobre posibles ataques a los recursos que prestan servicios en Internet (como los servidores Web o de correo) o ayudar a identificar los sectores de la red local que son más propicios para la propagación de virus y así tomar medidas preventivas como la capacitación de los usuarios.

Sin embargo, la efectividad de un IDS depende en gran medida del personal que lo utiliza (de su experiencia, habilidades y de los recursos que tengan disponibles para realizar su trabajo), aspecto que no se tratará con detalle en este documento.

5.2 Infraestructura de red de la Universidad del Cauca

La red está dividida en ocho sectores: Ingenierías, Medicina, VRI (Vice-Rectoría de Investigaciones) y Educación se encuentran al norte de la ciudad; El Carmen y Santo Domingo en el centro de la ciudad; Las Guacas en el área rural de Popayán y finalmente sede en Santander de Quilichao. Cada sector puede comprender uno o más edificios geográficamente cercanos. En cada edificio hay un centro de cableado principal y puede haber uno o varios centros de cableado secundarios, en los cuales se encuentran los equipos de red (switches, hubs, etc.).

En la **Figura 5.1** se muestra un diagrama de la red actual (enero de 2006) de la Universidad del Cauca.

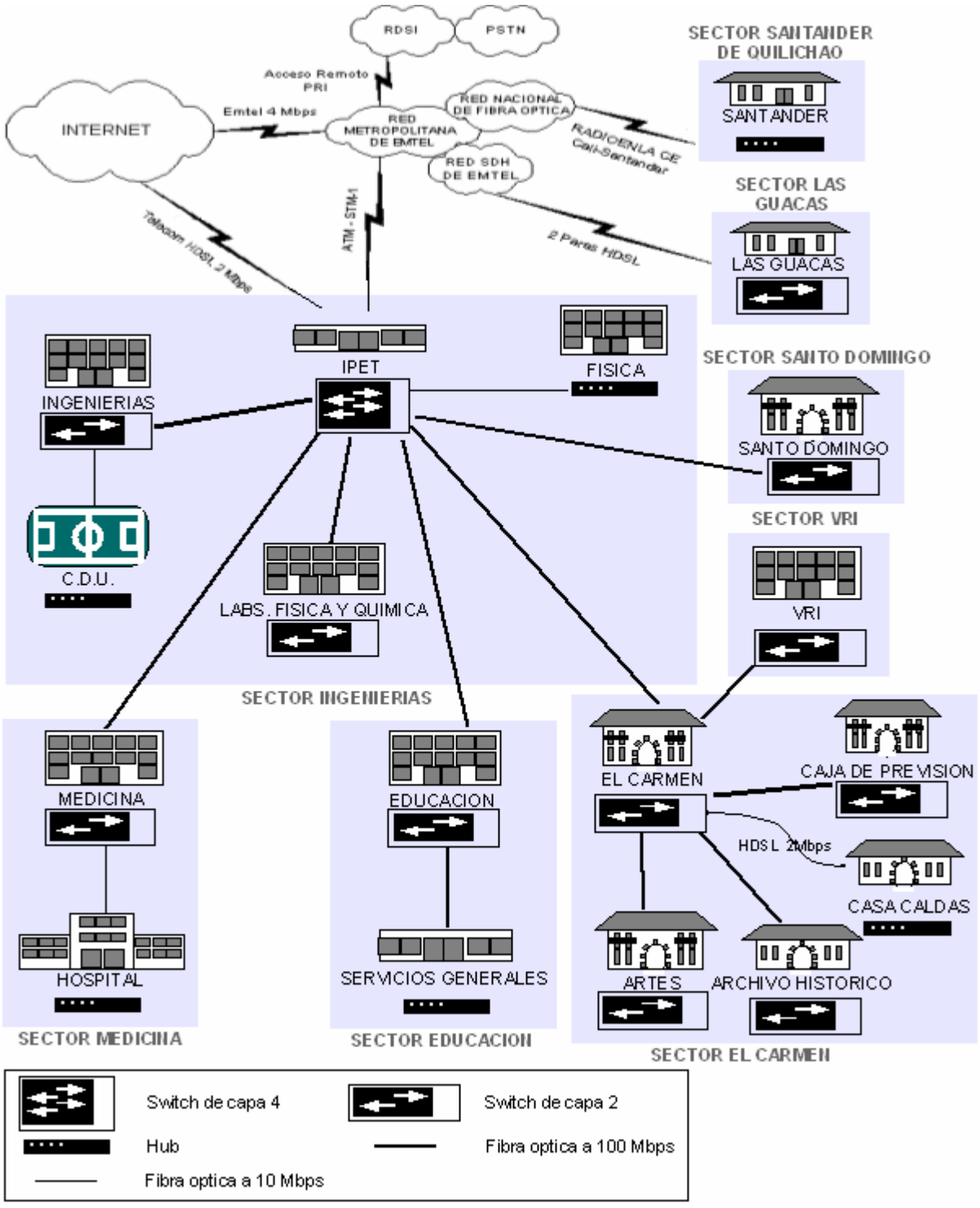


Figura 5.1. Red de Datos de la Universidad del Cauca (enero de 2006).

5.3 Identificación de los recursos críticos

En la siguiente tabla (**Tabla 5.1)** se muestra el resultado de un análisis parcial de riesgos tecnológicos.

Nombre de recurso	**Ubicación**	**Servicios que proporciona**	**Impacto para la organización en caso de un incidente de seguridad**	**Importancia para la organización**	**Personal del área usuaria que valida la información**
Servidores de área financiera	Zona de seguridad en red interna. División de sistemas, Edificio Educación	Almacena información contable, de nómina, matrículas financiaras.	Perdida de capacidad para pagos de nómina, control de matriculas financiaras.	ALTO	Personal División de sistemas. Personal del área financiara.
Servidores de calificaciones	Zona de seguridad en red interna. División de sistemas, Edificio Educación	Mantiene un registro centralizado de las calificaciones de los estudiantes.	Incrementa el riesgo de fraude en calificaciones. Retrasos en matriculas académicas y otros procesos académicos.	MEDIO	Personal División de sistemas. Secretarias de las distintas facultades.
Servidor Web que alberga el portal Web de la universidad	Zona desmilitarizada. Edificio IPET.	Publicación de información para la comunidad e interfaz Web para el servicio de correo electrónico	Afectación a la imagen de la universidad. Algunos usuarios no podrían utilizar el servicio de correo.	MEDIO	Administrador de contenidos. Comunidad universitaria.
Servidores de correo electrónico	Zona desmilitarizada. Edificio IPET.	Proporciona comunicación interna y externa. Herramienta para grupos de investigación.	Retrasos es procesos investigativos.	MEDIO	Administrador de Servicios de Internet. Comunidad universitaria.

Computadores de la red local asignados a profesores y administrativos.	Red interna. Toda la universidad.	Facilita la consulta de información investigativa y permite realizar labores administrativas varias.	Información investigativa menos actual. Retrasos en algunas labores administrativas. Posible propagación de virus	BAJO	Profesores y administrativos.
Computadores de la red local accesibles o asignados a estudiantes.	Red interna. Toda la universidad.	Facilita la consulta de información y realización de labores académicas.	Retrasos en elaboración de actividades académicas. Posible propagación de virus	BAJO	Estuduantes

Tabla 5.1. Recursos críticos.

El campo **personal del área usuaria que valida la información** se refiere a las personas que conocen mejor la funcionalidad e impacto al negocio del recurso. En un plan completo, este análisis debe ser revisado al final por un ejecutivo o grupo de trabajo que mantenga un nivel jerárquico superior al de todas las áreas mencionadas, para verificar que los resultados son consistentes en un contexto más general: el de toda la organización. En este plan parcial no se realizará dicho procedimiento.

Los recursos informáticos más importantes para la universidad se encuentran en la División de sistemas. Por cuestiones de seguridad, el análisis de estos recursos no es profundo y se mencionan a modo de ejemplo, el análisis se centra más en el área identificada como Zona Desmilitarizada o DMZ en donde se encuentran los recursos que dan servicios por Internet y en la red interna de la Universidad en general.

La Zona Desmilitarizada (DMZ) está compuesta por todos los equipos que (a diferencia de los de la red interna) son visibles desde Internet, entre ellos están los equipos que albergan sitios Web de grupos de investigación, los servidores que ofrecen servicios de correo y demás.

5.4 Perfil del personal encargado de la administración del NIDS

El perfil y experiencia del personal que utilizará el Sistema de Detección de Intrusos debe estar basado en las características y funciones de los sistemas más críticos de la organización (a los cuales se pretende proteger), y no en el conocimiento o experiencia en el manejo de un producto en particular. Sin embargo, es un hecho que existen requisitos mínimos que toda persona que desee entrar en este perfil, debe cumplir. Algunos de estos requisitos mínimos son: conceptos de seguridad informática, conocimiento del sistema operativo del recurso a proteger, redes y seguridad en redes, aplicaciones y seguridad en aplicaciones, principios éticos y profesionales.

En la red de datos de la Universidad del Cauca (con la organización actual), los monitores del área de servidores cumplen con casi todos los requisitos mínimos mencionados anteriormente. Sin embargo, añadir a las numerosas labores que tiene un monitor de dicha área actualmente, la administración de un NIDS, podría ocasionar que el sistema no sea aprovechado adecuadamente.

Se debería asignar personal de seguridad para vigilar los sistemas más críticos; no es necesario asignar personal que esté completamente dedicado a la vigilancia de un sólo sistema, pero es importante cubrir todos los sistemas que son imprescindibles para la operación de la organización. Como se vio en la **Tabla 5.1** los sistemas imprescindibles para la Universidad del Cauca se encuentran a cargo de la División de Sistemas, pero también reciben apoyo esporádico del equipo de la Red de Datos incluyendo a los monitores del área de servidores.

5.5 Selección del tipo, cantidad y arquitectura del NIDS

Para la implantación del Sistema de Detección de Intrusos de Red se deben considerar varios aspectos que se muestran a continuación:

5.5.1 Ubicaciones donde se instalarán los sensores

De acuerdo con los recursos críticos identificados anteriormente, se definen varios puntos en los cuales poner sensores NIDS. El punto en el que la detección de intrusiones tendría más utilidad es la División de Sistemas, sin embargo no se tiene información suficiente para definir aspectos técnicos de implantación del sensor en ese sector.

Descartando ese sector, se definen tres puntos en los cuales la detección de intrusiones tendría efectos benéficos en el mejoramiento de la seguridad y disponibilidad de los servicios informáticos de la universidad, sin hacer grandes inversiones en hardware (ver **Tabla 5.2**).

ID del sensor	Ubicación	Utilidad
S1	Zona desmilitarizada. Edificio IPET	Este sensor estaría enfocado a la detección de posibles ataques a los servicios Web, de correo electrónico, de FTP y DNS.
S2	Red interna, en el switch principal. Edificio IPET	Vigilaría las comunicaciones entre los edificios del sector de Ingenierías, Medicina, Educación, El Carmen y Santo Domingo para tener un mayor control del lugar donde se inician las propagaciones de virus o spyware en la red de la universidad, así como el análisis de comportamientos sospechosos por parte de los usuarios.
S3	Red interna. Edificio El Carmen.	Igual que el S2, pero enfocado en las facultades y edificios del centro de la ciudad.

Tabla 5.2. Sensores a instalar.

5.5.2 Arquitectura de administración

Por la poca cantidad de sensores instalados, se podría pensar que una arquitectura de sensores individuales sería la más óptima, sin embargo por la separación geográfica del sensor S3 se recomienda una arquitectura a dos capas con una consola de administración centralizada (ver **Figura 5.2**).

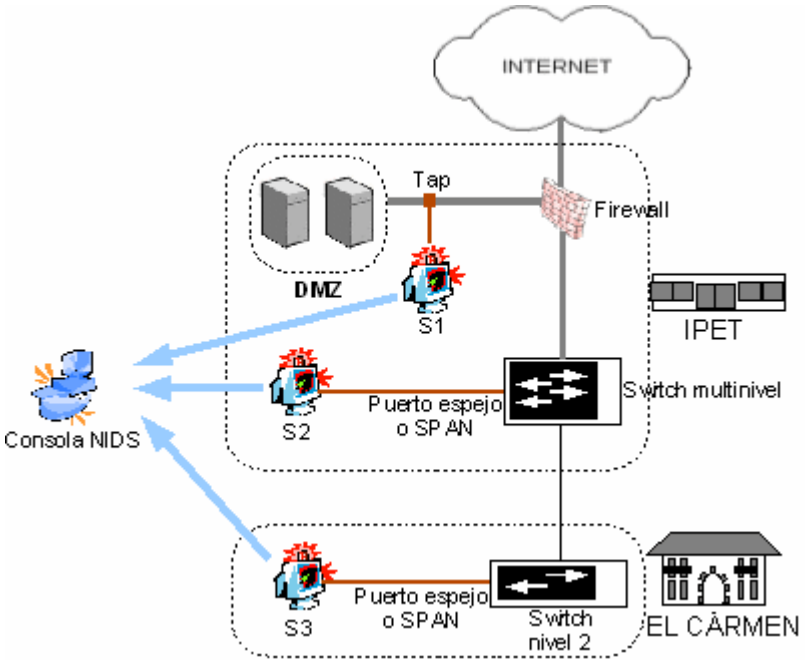


Figura 5.2. Ubicación de los sensores del NIDS.

5.6 Aspectos técnicos

A la hora de implantar un Sistema de Detección de Intrusos de Red se debe definir aspectos como el software que se utilizará, los puntos de conexión y las medidas para vigilar el tráfico encriptado que manejan los recursos que se quiere proteger.

5.6.1 Herramientas software a utilizar

La herramienta de software libre Snort se puede considerar como un simple sensor, pero gracias a las contribuciones de la comunidad se cuenta con programas útiles para montar una solución de detección de intrusos completa, entre ellos están:

- **BASE:** La consola de análisis de alertas BASE (*Basic Analysis and Secutiry Engine* – Motor Básico de Análisis y Seguridad) permite manejar las alertas y los logs generados por distintos sensores, que han sido guardados en una base de datos centralizada. Es una aplicación Web que requiere la instalación de un servidor Web (por ejemplo Apache) y el sistema de base de datos MySql.

- **SnortCenter:** Es una aplicación Web que tiene los mismos requisitos que el anterior. Sirve para administrar diferentes sensores localizados remotamente, permite alterar su configuración, modificar las reglas de detección y habilitar módulos, entre otras cosas. Para lograr esto se debe instalar en el sensor, que ya tiene instalado Snort, un programa conocido como el *SnortCenter Agent*. Este vigila a Snort y reporta a la consola central de SnortCenter, también acepta instrucciones de la consola central y realiza los cambios apropiados en la configuración de Snort.

- **Barnyard:** Esta herramienta mejora notablemente el rendimiento de Snort. Una de las actividades que más trabajo le cuesta a Snort es el registro de alertas o logs. Los datos necesitan ser recolectados, formateados y escritos. En el caso de la escritura en una base de datos, Snort debe escribir la alerta y esperar la confirmación de que fue escrita correctamente.

  Snort tiene la habilidad de guardar las alertas en un formato binario. Esto lo hace muy rápido, debido a que no tiene que realizar procesamiento sobre los datos. La aplicación Barnyard lee este archivo, formatea los datos y los escribe con el

mecanismo de salida escogido. Este mecanismo puede ser una base de datos para que luego sean leídos con BASE.

- **Oinkmaster:** Es el sistema de actualización de Snort por excelencia. Esta compuesto por una serie de scripts hechos en Perl que crean una copia de seguridad del directorio con las reglas antiguas, descarga las últimas, realiza cambios de configuración y genera un reporte de los cambios en las nuevas reglas comparadas con las del directorio antiguo. Se recomienda configurar esta aplicación para que sólo muestre los cambios en las nuevas reglas y no los aplique, debido a que cada organización tiene reglas personalizadas de acuerdo a sus necesidades y cada actualización debe ser analizada por el administrador.

Estas herramientas estarían distribuidas de la siguiente forma:

- **Consola NIDS:** Tendría instalado: MySql, Apache, las aplicaciones Web BASE y SnortCenter, y Oinkmaster para ayudar al administrador a seleccionar las actualizaciones del sistema. Adicionalmente requiere de un navegador Web para utilizar las consolas de análisis y administración localmente.
- **Sensores:** Cada sensor tendría instalado: Snort, el SnortCenter Agent y Barnyard para mejorar el rendimiento, sobre todo en los sensores conectados a un puerto espejo o SPAN.

El hecho de utilizar aplicaciones Web en la consola NIDS hace que el administrador pueda acceder a estos recursos desde cualquier PC con un navegador Web, convirtiendo opcionalmente la arquitectura de administración en una de 3 capas.

5.6.2 Consideraciones sobre los taps y conexiones en puerto espejo o SPAN

Un *tap* es un dispositivo que crea una copia del tráfico que viaja por un cable de red, ya sea cobre o fibra óptica. Algunos son comerciales y otros pueden ser construidos manualmente. El problema con el uso de este tipo de dispositivos es que representan un único punto potencial de falla para el cable en el que está instalado, por eso se debe hacer constante vigilancia de estos dispositivos.

Las conexiones en puerto espejo de un switch, también llamadas por Cisco SPAN (*Switched Port ANalyser*), hacen posible que dicho dispositivo realice una copia del tráfico de uno o varios puertos y la envíe a un puerto designado en el cual se puede conectar el sensor NIDS. Se debe tener cuidado con este tipo de conexiones, ya que si el ancho de banda total de todos los puertos que se quieren vigilar es mayor que el ancho de banda del puerto en el cual se tiene conectado el sensor, el tráfico extra se perderá y no será monitoreado.

Para prevenir esto se recomienda que los sensores S2 y S3 estén conectados con una interfaz Gigabit Ethernet (1000 Mbps). De esta forma, aún con las condiciones de tráfico mas críticas, que sería en el switch multinivel todos los enlaces a su máximo (6 x 100Mbps + 10Mbps = 610 Mbps), no se superaría la capacidad de la interfaz.

5.6.3 Consideraciones sobre el tráfico encriptado

En el caso del servidor Web que presta el servicio de acceso al correo electrónico por medio de una conexión segura con encriptación SSL, el sensor NIDS no podría monitorear este tráfico encriptado, de modo que ataques a nivel de aplicación (como la inyección de instrucciones SQL) no serían detectados (**Figura 5.3**).

Figura 5.3. El sensor no puede vigilar tráfico encriptado.

Para evitar este inconveniente se puede adoptar el uso de un Proxy SSL, también conocido como Switch de contenido o Acelerador SSL. Este dispositivo se coloca entre el cliente y el servidor manejando las labores de encriptación. El tráfico entre el servidor Web y el Proxy SSL no está encriptado, pero el tráfico entre el proxy y el cliente Web si. Conectando el sensor NIDS entre el servidor Web y el proxy se logra que el tráfico sea monitoreado (**Figura 5.4**).

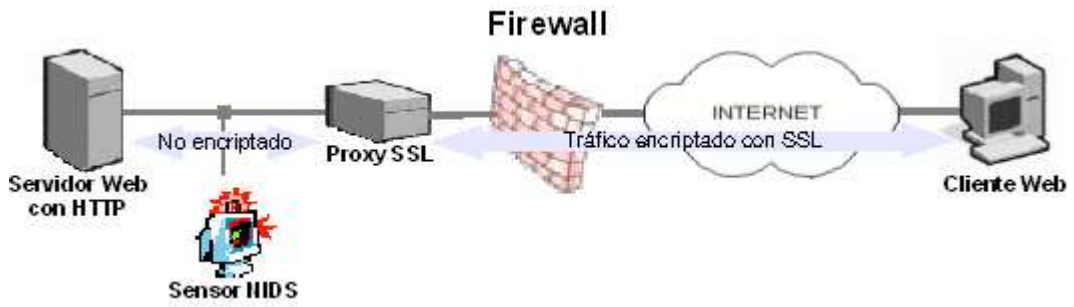


Figura 5.4. Con esta configuración el tráfico si puede ser vigilado.

CONCLUSIONES Y RECOMENDACIONES

- La respuesta a la pregunta "¿Es verdad que las tecnologías adaptativas de IA mejorarán de alguna forma a los IDS?" es:

  Si, la utilización de este tipo de tecnologías facilitará la administración de los IDS, debido a que simplificarán el manejo de las reglas reduciendo su número y complejidad, adicionalmente se adaptan a las características de cada red permitiendo crear un perfil que ayudará a la detección de variantes desconocidas de ataques.

  De acuerdo con los resultados de las pruebas presentados en este proyecto, se ven prometedoras mejoras en estas aplicaciones gracias al uso de tecnologías de IA, como: un mejor rendimiento por la reducción en el consumo de memoria, debido a la eliminación de las reglas lógicas; y disminución del número de falsos negativos. En posteriores pruebas se puede lograr la disminución del número de falsos positivos con un ajuste adecuado de los parámetros de configuración como el nivel de sensibilidad. Desde el punto de vista del desarrollador, el uso de tecnologías de IA como las Redes Neuronales, facilita en gran medida el diseño e implementación de las aplicaciones IDS.

- Las RNA's facilitan el trabajo a los desarrolladores y reducen en gran medida la complejidad de los algoritmos. La detección de escaneo de puertos sin la RNA implicaría el uso de una cantidad mucho mayor de líneas de código. Naturalmente, se debe señalar que hay una tarea de filtraje de la información del medio que permite la codificación de entradas, pero es relativamente simple comparada con la detección del ataque en sí por medio de procedimientos algorítmicos.

- Las RNA's no se prestan para desarrollar soluciones genéricas a problemas, estas se desempeñan bien en problemas puntuales. Esto se ve reflejado en el hecho de que la RNA implementada sólo sirve para detectar escaneos de puertos, si se quisiera detectar otro tipo de ataque se tendría que implementar otra red y realizar un nuevo análisis para la codificación de las entradas, de modo que sea apropiada para dicho tipo de ataque.

- La detección de ataques a nivel de aplicación, como por ejemplo shellcodes, es una tarea que puede llegar a ser significativamente más difícil que la detección de ataques a nivel de red y transporte, como escaneo de puertos, debido entre otras cosas al hecho de que la información puede haber pasado por un proceso de encriptación y a la misma complejidad del ataque, que puede incluir diferentes tipos de patrones dependiendo del sistema operativo de la víctima o de lo que se quiera desencadenar en ella.

- El trabajo colaborativo sobre herramientas de software libre tiene inconvenientes no completamente insalvables, pero sí de cierta consideración, que generalmente tienen que ver con problemas de comunicación y de sincronización. Sistemas de control de versiones como CVS han hecho más fácil el acceso a las versiones de desarrollo, sin embargo, eso no mitiga del todo el hecho de que el grupo de trabajo está distribuido por todo el mundo y que en muchos de los casos no son desarrolladores de tiempo completo para el proyecto.

- Una de las mejores ventajas del software libre es que por medio del análisis del código se puede aprender muchas nuevas formas de abordar un problema, tanto desde el punto de vista meramente técnico como desde el de ingeniería. Viéndolo así, puede ser una gran herramienta dentro de un programa académico, para que los estudiantes mejoren sus habilidades a partir del trabajo libre que han realizado programadores más experimentados.

- La elección de las entradas de una red neuronal es definitiva para lograr la convergencia hacia el nivel de error deseado durante el entrenamiento. Una elección errónea puede generar unos niveles de error inaceptables, largos tiempos de entrenamiento o se deben proporcionar sets de entrenamiento redundantes o innecesariamente complicados.

- En la mayoría de los casos el proceso de validación e integración de una red neuronal es un proceso de ensayo y error, hasta que se encuentra las funciones de activación y aprendizaje adecuadas, la cantidad de las capas y neuronas conveniente, e incluso los pesos iniciales con que debe entrenarse la red al principio.

- En un programa que pude estar corriendo constantemente durante prolongados períodos de tiempo, el uso de memoria es un factor que debe ser vigilado con mucho cuidado, para no comprometer la disponibilidad del sistema o afectar el correcto funcionamiento de otras aplicaciones.

- Para un detector de escaneos 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 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.

- La implantación de un NIDS en la red de la Universidad del Cauca podría mejorar la respuesta a incidentes como la propagación de virus, que en ocasiones anteriores

han provocado dolores de cabeza a profesores y administrativos, así como reducir el impacto provocado por estos. Pero no es un proceso sencillo y se requiere de un plan organizado que considere los aspectos técnicos, de personal y los procedimientos para respuesta a incidentes.

- Cuando quiera realizar un proyecto sobre una aplicación, esté en constante contacto con la comunidad de desarrolladores o de usuarios de la aplicación, suscríbase a listas de correos y de acceso a foros. No sólo para enterarse de las inquietudes y problemas experimentados por ellos, sino para compartir las propias y colaborar con los que apenas se integran al proyecto. Una buena estrategia es estar pendiente de los *call for papers* de las diferentes revistas *on-line* y escribir artículos, de esta forma otros se enterarán del trabajo que se está desarrollando y se puedan coordinar acciones para complementar las actividades que se planea realizar.

- Documentar los resultados a medida que se van produciendo, incluyendo fracasos, además de organizar adecuadamente esas notas. De esta forma, al finalizar el proyecto será más simple generar la documentación final más efectiva y rápidamente.

GLOSARIO

Benchmark: Comparación entre distintas marcas o productos para identificar sus fortalezas y debilidades.

BoF: Buffer overFlow. Un tipo de vulnerabilidad que permite ejecución remota de código.

DoS: Denial of Service – Ataque de Denegación de Servicio.

IA: Inteligencia Artificial.

IDS: Intrusion Detection System – Sistema de Detección de Intrusiones.

IPS: Intrusion Prevention System – Sistema de Prevención de Intrusiones

IPv4: Protocolo de Internet versión 4

NIDS: Network IDS – IDS de Red.

RNA: Red(es) Neuronal(es) Artificial(es).

Spyware: Software espía. Software malicioso que recolecta información del equipo en el que se ejecuta y la envía a un tercero por Internet.

BIBLIOGRAFÍA

[Arboleda & Bedón 2005] Arboleda, Andrés & Bedón, Charles (2005). *Snort diagrams for developers*. http://www.snort.org/docs/#devel

[Bace & Mell 2000] Bace, Rebecca & Mell, Peter (2000). *Intrusion Detection Systems*. NIST Special Publication.

[Bedón & Arboleda 2005] Bedón, Charles & Arboleda, Andrés (2005). *Snort preprocessors development kickstart*. http://www.snort.org/docs/#devel

[Bigus 2001] Bigus, Joseph P. & Bigus, Jennifer (2001). *Constructing intelligent agents using JAVA.* New York: John Wiley & Sons.

[Biv02] 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.

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

[Bon98] Bonifacio Jr., J. M., Cansian, A. M., Carvalho, A. C. P. L. F., and Moreira, E. S. (1998a). Neural Networks Applied in Intrusion Detection Systems. In Proceedings of the IEEE International Joint Conference on Neural Networks(IJCNN'98).

[Can97] Cansian, A. M., S. Moreira, E., Carvalho, A. C. P. L. F., and Bonifácio Jr, J. M. (1997). Network Intrusion Detection using Neural Networks. In Proceedings of the

International Conference on Computational Intelligence and Multimedia Applications (ICCMA'97), pages 276-280.

[Cann98] Cannady, J. (1998). Neural Networks for Misuse Detection: Initial Results. Proceedings of the Recent Advances in Intrusion Detection '98 Conference, 31-47.

[Castillo et al. 1997] Castillo, E., Gutierrez, J., Hadi, A. (1997). *Sistemas expertos y modelos de redes probabilísitcas*. España: Universidad de Cantabria.

[Chavan et al. 2004] Chavan, S., Shah, K., Dave, N., Mukherjee, S. (2004). Adaptive Neuro-Fuzzy Intrusion Detection Systems. *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'04).* IEEE.

[Cox & Gerg 2004] Cox, Kerry & Gerg, Christopher (2004). *Managing Security with Snort and IDS Tools*. O'Reilly.

[D. M. 2004] Death Master (2004). *Sabuesos en la Red: El escaneo de puertos*. http://www.death-master.tk/.

[DAR04] DARPA Intrusion Detection Evaluation, Lincoln Laboratory, Massachusetts Institute of Technology. http://www.ll.mit.edu/IST/ideval/pubs/pubs_index.html, (07/10/2004).

[Deb92a] H. Debar and Dorizzi, B. An Application of a Recurrent Network to an Intrusion Detection System. In IEEE, editor, International Joint Conference on Neural Networks 1992, pages 478-483.

[Deb92b] H. Debar, M Becker, D. Siboni. A Neural Network Component for an Intrusion DetectionSystem. Proceedings, IEEE Symposium on Research in Computer Security and Privacy, 1992. pp 240-250.

[Desai 2003] Desai, Neil. (2003). *Intrusion Prevention Systems: the Next Step in the Evolution of IDS*. SecurityFocus. http://www.securityfocus.com/infocus/1670

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

[End98] D. Endler. Intrusion detection: Applying machine learning to solaris audit data. In Proceedings of the 1998 Annual Computer Security Applications Conference (ACSAC'98), pages 268--279, Los Alamitos, CA, December 1998. IEEE Computer Society, IEEE Computer Society Press. Scottsdale, AZ.

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

[Gho99] Ghosh, A. and Schwartzbard, A. (1999a). A Study in Using Neural Networks for Anomaly and Misuse Detection. In Proceedings of the 8th USENIX Security Symposium (SEC'99).

[Herrera 2003] Herrera, Omar. (2003). *Implementación y Configuración de Sistemas de Detección de Intrusos.* CISSP.

[Hög98] Höglund, A., Hätönen, K., and Tuononen, T. (1998). A UNIX Anomaly Detection System using Self-Organising Maps. Web proceedings of the First International Workshop on Recent Advances in Intrusion Detection (RAID'98).

[Jordan 1986] Jordan, M. I. (1986). Serial order: a parallel distributed processing approach. *Institute for Cognitive Science Report 8604*. San Diego: University of California.

[Ken98] K. Kendall, A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems, Massachusetts Institute of Technology Master's Thesis, 1998.

[Lip99] Lippmann, R. P. and Cunningham, R. K. (1999). Improving Intrusion Detection Performance using Keyword Selection and Neural Networks. Web proceedings of the 2nd International Workshop on Recent Advances in Intrusion Detection (RAID'99).

[Mar01] D. Marchette, Computer Intrusion Detection and Network Monitoring, A Statistical Viewpoint. New York, Springer, 2001.

[Martín & Sanz 2001] Martín del Brío, Bonifacio & Sanz Molina, Alfredo (2001). *Redes Neuronales y Sistemas Difusos.* España: Ed. Alfaomega.

[Morales 2002] Morales, G. (2002). *Introducción a la lógica difusa.* México: Centro de Investigación y Estudios Avanzados del IPN.

[Newell 1980] Newell, A (1980). Physical symbol systems. *Cognitive Science* 4:185-203.

[Noonan 2005] Noonan, Wes. (2005). *Intrusion interrupted*. Articulo en Redmond, http://redmondmag.com.

[Ortega 2004] Ortega, Urko. (2004). *Estado del arte: Sistemas de Detección de Intrusos*. España: Mondragon Unibertsitatea.

[Ram03] 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.

[Rho00] Rhodes, B. C., Mahaffey, J. A., and Cannady, J. D. Multiple Self-Organizing Maps for Intrusion Detection. In Proceedings of the 23rd National Information Systems Security Conference (NISSC 2000).

[Rich & Knight 1991] Rich, E. & Knight, K. (1991). *Artificial Intelligence.* New York: McGraw-Hill.

[Russell & Norvig 1995] Russell, S. & Norvig, P. (1995). *Artificial Intelligence: A Modern Approach.* Englewood Cliffs, NJ: Prentice Hall.

[Rya98] Jake Ryan, Meng-Jang Lin, and Risto Miikkulainen. Intrusion Detection with Neural Networks. In Advances in Neural Information Processing Systems 10 (Proceedings of NIPS'97, Denver, CO). Cambridge, MA: MIT Press, 1998.

[SCMagazine 2005] *Group Test: Intrusion Prevention (2005)*. SC Magazine US. http://www.scmagazine.com/us/grouptest.

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

[Zadeh 1988] Zadeh, L. (1988). Fuzzy logic. *IEEE Computer*, 1:83.

[Zell et al. 1995] Zell, A., Mamier, G., Vogt, M., Mache, N. (1995). *SNNS, Stuttgart Neural Network Simulator: User manual, version 4.1*. University of Stuttgart.

ANEXO 1: MANUAL DE USUARIO

PortscanAI es un preprocesador (una clase de plug-in) para el Sistema de Detección de Intrusos en Redes (NIDS) Snort. PortscanAI ayuda a la detección de escaneos de puertos de diversas clases, utilizando una Red Neuronal Artificial para la identificación de este tipo de ataques.

En el CD del proyecto se encuentran las carpetas 'Analysis_console' y 'source'. La primera posee los archivos .php de la aplicación Web que sirve para analizar la información arrojada por el preprocesador PortscanAI. La segunda carpeta contiene el código fuente del preprocesador PortscanAI, este debe ser unido al código de Snort y así compilar todo como una sola aplicación.

PortscanAI ha sido probado únicamente en Linux. Se recomienda leer el manual de Snort ya que PortscanAI es sólo una parte de dicho Sistema de Detección de Intrusos.

Instalación del preprocesador

Antes de instalar PortscanAI debe descargar el código fuente de Snort (www.snort.org/dl/) y descomprimirlo en una carpeta que de ahora en adelante se identificará como “carpeta de instalación de Snort”.

- En la carpeta 'source' se encuentra el archivo 'PortscanAI.zip', copielo en la carpeta de instalación de Snort (Por ejemplo: /home/user/snort-2.4.1) y descoprimalo.

- Dentro de la carpeta de instalación de Snort se habrá creado una carpeta llamada 'PortscanAI' y en el interior de esta se encuentra un archivo de script llamado 'src_install.sh'. Ejecute el script para unir el código fuente de PortscanAI con el de la instalación de Snort. Tenga en cuenta que el script sobreescribe los siguientes archivos:

etc/snort.conf,
src/plugbase.c,
src/preprocessors/Makefile.in
src/preprocessors/flow/flow_callback.c

- Ejecute './configure' en la carpeta de instalación de Snort y luego ejecute 'make' para compilar todo el conjunto.

Ejecución del preprocesador

Para utilizar el preprocesador PortscanAI se debe habilitar y configurar en el archivo 'snort.conf' (leer el manual de Snort). El archivo 'snort.conf' que sobreescribe al original luego de ejecutar el script de instalación, ya tiene las lineas necesarias para habilitar a PortscanAI y viene con una configuración por defecto.

PortscanAI se apoya en el preprocesador flow, por eso este debe estar habilitado en el archivo de configuración (snort.conf) si usted desea utilizar PortscanAI.

Se tienen las siguientes opciones de configuración:

ignorebc <1|0> - El preprocesador ignora el tráfico de broadcasr de la red (1) o los analiza (0).

rna_output <number> - Esta opción es utilizada para la generación de sets de entrenamiento para la red neuronal y no tiene ninguna utilidad en esta versión del preprocesador.

analyze_thr_lower <number> - A partir de qué cantidad de peticiones de conexión detectadas por dirección IP se inicia el análisis.

analyze_thr_upper <number> - En qué cantidad de peticiones de conexión detectadas por dirección IP se reinicia el análisis. Indica el fin de la "ventana".

sense_level <0 ... 0.1> - Nivel de sensibilidad. Por ejemplo: 0.01 es baja sensibilidad y 0.1 la máxima sensibilidad. Se recomienda colocar 0.05.

net_topology <0|1> - Define el modelo de red neuronal a ser utilizada por el preprocesador. Por ahora PortscanAI trae entrenados dos modelos de redes neuronales: Multi-Layer Perceptron (MLP) y una red neuronal Elman. Las opciones reconocidas son: 0 para una red MLP, 1 para una ELMAN.

Instalación de la consola de análisis

Requerimientos:

Servidor Web instalado (probado con Apache2)
MySql instalado
Librería jpgraph para PHP

1. Para utilizar esta consola se requiere que la base de datos en MySql con el nombre portscanai esté creada. Para esto hacer lo siguiente:

   MySql debe estar instalado. Para crear la base de datos puede utilizar el archivo 'portscanai.sql' con el siguiente comando:

   mysql –u<usuario> -p portscanai < portscanai.sql

2. La librería de PHP 'jpgraph' debe estar instalada. Para esto copie el código de la librería en el directorio raíz del servidor Web.

3. Configure los parámetros de la base de datos (host, usuario, contraseña) en el archivo 'spp_ai_log_parser.php' y la ruta a jpgraph en el archivo 'graph.php'.

4. Snort debe ser compilado para MySql:

```
./configure --with-mysql
make
```

Uso de la consola de análisis

La consola de análisis Web muestra el contenido de las tablas Hash internas del preprocesdor, las cuales contienen información que puede ser útil para el administrador. Las tablas tienen la siguiente apariencia:

Hash de IPs

IP	hits_as_src	hits_as_dst	av_rcv_time	av_snd_time	negative_resp	rst_resp	win_count
10.1.13.18	1	0	0	0	0	0	0
10.1.13.19	1	0	0	0	0	0	0
10.1.13.20	1	0	0	0	0	0	0
10.1.13.21	1	0	0	0	0	0	0
10.1.13.22	1	0	0	0	0	0	0
10.1.13.221	5	0	0	12.0497	0	0	0
10.1.13.38	0	64	0.000685	0	121	0	1
10.200.1.130	0	72	0.00023	0	109	0	1
10.200.2.112	0	1	0	0	0	0	0
10.200.2.114	2	0	0	0.0005	0	0	0
10.200.2.119	1	0	0	0	0	0	0
10.200.2.136	4	0	0	9.15786	0	0	0

Hash de Relaciones Directas

IP src	IP dst	Numero de Hits
10.1.13.38	172.16.130.87	1
10.200.1.130	172.16.130.195	1
10.200.1.130	172.16.130.87	1
10.200.2.112	172.16.42.41	1
10.200.2.173	172.16.130.127	1
10.200.2.173	172.16.130.169	1
10.200.2.174	172.16.130.127	1
10.200.2.174	172.16.130.169	6
10.200.2.174	172.16.130.183	1
10.200.2.174	172.16.130.195	1
10.200.2.32	194.83.71.244	1
10.200.2.32	200.115.114.30	1
10.200.2.32	201.34.232.26	1
10.200.2.32	202.74.220.123	1
10.200.2.32	208.195.210.141	43

En la tabla Hash de IPs se muestran los siguientes datos por cada dirección IP capturada:

- **hits_as_dst** (*hits as destination* – *hits* como destino): número de paquetes de inicio de conexión (*flows*) 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.

- **hits_as_src** (*hits as source* – *hits* como origen): número de paquetes de inicio de conexión (*flows*) 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.

- **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 escanedo.

- **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.

- **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

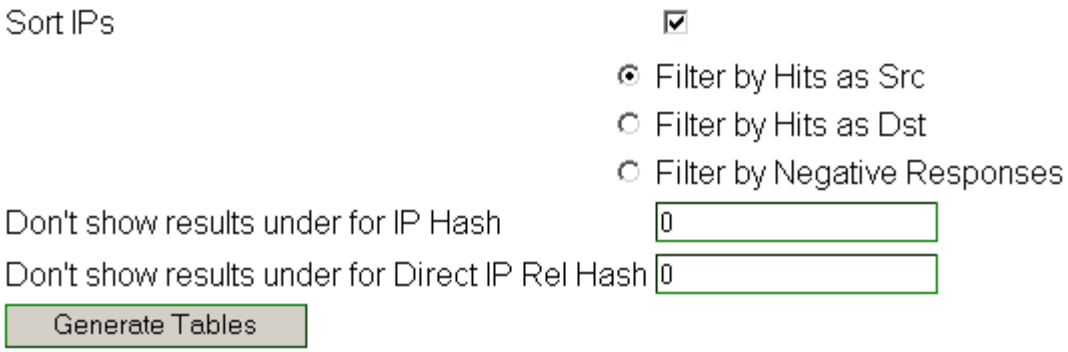
negativas para el protocolo TCP están dadas generalmente por el envío de un paquete con las banderas RST y ACK fijadas. Un valor alto puede indicar que esta dirección IP está siendo escaneada, ya que no es normal que un host reciba muchas peticiones en sus puertos cerrados.

Las columnas **rst_resp** y **win_count** significan: el número de paquetes con la bandera RST enviados por la dirección IP y el número de ventanas que han transcurrido (cuantas veces se han reiniciado las mediciones), respectivamente.

En la tabla Hash de Relaciones Directas se muestra el siguiente dato por cada dirección IP capturada:

- **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.

En la página de la consola también se dispone de controles para filtrar y organizar los datos de las tablas, como se ve a continuación:



Marque la casilla de verificación “Sort IPs“ si desea que las direcciones IP en las tablas sean organizadas en orden ascendente.

En el campo “Don’t show results under for IP Hash” coloque el valor de filtrado para la tabla Hash de IPs. Las direcciones IP cuyo valor en la columna seleccionada sea menor que el valor de filtrado no serán mostradas. La selección de la columna se realiza con los comandos de selección múltiple:

“Filter by Hits as Src”: para utilizar la columna **hits_as_src**.
“Filter by Hits as Dst”: para utilizar la columna **hits_as_dst**.
“Filter by Negative Responses”: para utilizar la columna **negative_resp**.

En el campo “Don’t show results under for Direct IP Rel Hash” coloque el valor mínimo para filtrar los pares de direcciones IP mostradas en la tabla Hash de Relaciones Directas.

Presione “Generate Tables” para generar las tablas con las opciones de filtrado.