APLICACIÓN DE REDES NEURONALES PARA LA DETECCIÓN DE INTRUSOS EN REDES Y SISTEMAS DE INFORMACIÓN

RESUMEN
Exponer de forma clara y precisa los conceptos de las redes neuronales y de los Sistemas de Detección de Intrusos (IDS) y estudiar si la integración de estas dos tecnologías puede generar una solución al problema de ataques a las redes de información. Además se analizan las ventajas y desventajas que tiene esta aproximación sobre los sistemas tradicionales para tratar de cubrir las actuales falencias y reforzar las características de estos.

PALABRAS CLAVES: Redes neuronales, IDS, ataque, falso positivo, falso negativo, intruso.

ABSTRACT
The project aims to expose in a brief and clear way the concepts of the neuronal networks and the Itrusion Detection Systems (IDS) and to study if the integration of these two technologies can generate a solution to the problem of network attacks. In addition the advantages and disadvantages this approach has on the traditional systems will be studied to try to cover the present weaknesses and to reinforce these systems characteristics.

KEYWORDS: *Neural networks, IDS, attack, false positive, false negative, intruder.*

CARLOS ALFONSO PÉREZ RIVERA
Ingeniero de Sistemas y Computación
Universidad Tecnológica de Pereira
cperez@utp.edu.co

JAIME ANDRES BRITTO MONTOYA
Ingeniero de Sistemas y Computación
Universidad Tecnológica de Pereira
jbrito@utp.edu.co

GUSTAVO ADOLFO ISAZA ECHEVERRY
Ingeniero de Sistemas, Esp. Software para Redes
Profesor de planta Departamento de Sistemas e Informática - Facultad de Ingeniería
Universidad de Caldas
gustavo.isaza@ucaldas.edu.co

1. INTRODUCCIÓN

¿Existe una mejora sustancial al construir un sistema de detección de intrusos aplicando redes neuronales?

Los sistemas de detección de intrusos son herramientas que han sido diseñadas para aumentar la seguridad de una red de datos. Los estudios realizados en esta área tuvieron su origen en el análisis de los flujos de datos de los sistemas vulnerados, inicialmente personal experto en el tema dedicó todo su tiempo a identificar el origen de las anomalías y evitar que afectara de nuevo al sistema de información, esto evolucionó en lo que se conoce como sistemas de detección basados en reglas, que han sido estándar hasta la actualidad.

Los sistemas basados en reglas requieren actualización permanente ya que si un ataque es modificado levemente el sistema es incapaz de detectarlo. Por esto la inteligencia artificial se ha planteado como una solución a las limitaciones de los sistemas tradicionales.

En este artículo se presenta una investigación que proporciona como resultado unas métricas de desempeño que dan una visión más amplia de la participación de la inteligencia artificial, en especial las redes neuronales, en el campo de la seguridad en redes informáticas. Así mismo se plantea y comprueba una nueva aproximación que implica un avance en el desarrollo de herramientas en esta área.

Se mostrarán los pasos que fueron necesarios para obtener un sistema de detección de intrusos basado en redes neuronales.

2. LA BASE DE CONOCIMIENTO

Era necesario en primer lugar obtener una base de conocimiento sobre la cual se pudiera trabajar, para esto se utilizó SNORT, un IDS muy conocido y de libre distribución, la batería de ataques NESSUS, el analizador de resultados con interfaz Web ACID, fundamentado sobre MySQL. Con estas herramientas se realizaron varias sesiones de ataques, almacenando los paquetes obtenidos (señalados como peligrosos por el Sistema de Detección de Intrusos - SNORT) en la base de datos de ACID, así mismo se almacenaron archivos de tráfico normal.

Los paquetes de tráfico normal fueron almacenados en formato TCPDUMP a diferencia de los peligrosos que fueron almacenados en la base de datos MySQL, esto debido a las características de detección de SNORT en conjunción con ACID.

Fecha de recepción: 27 Enero de 2005
Fecha de aceptación: 11 Abril de 2005

Se desarrollaron dos pequeñas aplicaciones, la primera realizada en PHP, la cual es capaz de leer las bases de datos de ACID donde se encontraban los paquetes peligrosos, así mismo, los datos de encabezados y del contenido eran exportados a un archivo separado por comas. Estos archivos fueron importados en un procesador de hoja de cálculo para su estudio.

Los datos de encabezado, que no variaban en todos los paquetes analizados, así como los que no influían en la detección (según las reglas de detección de SNORT) fueron descartados como no significativos, sin embargo algunos encabezados no fueron descartados para permitir futura compatibilidad al ampliar el universo de análisis, quedando seleccionados los siguientes 9 encabezados:

Tos (Tipo de Servicio): Indica una serie de parámetros sobre la calidad de servicio deseada durante el tránsito por una red.
Len (Longitud Total): Es el tamaño total, en octetos, del datagrama, incluyendo el tamaño de la cabecera y el de los datos.
Ttl (Tiempo de Vida TTL): Indica el máximo número de segundos que un paquete puede estar circulando.
Sport (Puerto Origen): Puerto TCP de la máquina emisora de datos.
Dport (Puerto Destino): Puerto TCP de la máquina destino.
Seq (Número de secuencia): Identifica el primer byte dentro de ese segmento de la secuencia de bytes enviados hasta ese momento.
Ack (Numero de reconocimiento): Contiene el próximo número de secuencia que el transmisor del ACK espera recibir.
Tcpflags (Banderas del protocolo): Este campo describe cual es el contenido del segmento.
Win (Tamaño de ventana): Tamaño de la ventana advertida por el receptor al transmisor (Sliding Window).

Se analizó de la misma manera los contenidos de los paquetes. El paquete detectado como peligroso de mayor longitud de contenido era de 393 caracteres, por lo tanto suficiente para realizar una detección exitosa dentro de nuestro universo.

La segunda aplicación, elaborada en C++, permite leer desde el archivo en formato TCPDUMP donde se encuentran almacenados los paquetes de tráfico normal. Este programa permite de igual manera filtrar los paquetes, seleccionar los encabezados significativos y cortar los contenidos al tamaño máximo identificado (393 caracteres de contenido) y finalmente almacenar estos resultados en un archivo de texto separado por comas.

3. NORMALIZACIÓN DE LA BASE DE CONOCIMIENTO

Una vez seleccionados los datos significativos, estos deben ser normalizados antes de ingresar en la red neuronal. Cada dato se dividió entre el número mayor posible para el campo en cuestión, de esta manera cada campo obtiene un valor entre 0 y 1 siendo homogéneo y correcto para su ingreso en la red neuronal. Esta normalización fue llevada a cabo por las dos pequeñas aplicaciones construidas, de la siguiente manera:

Campo	Longitud (bits)	Dividido entre
Tos	8	255
Len	16	65535
Ttl	8	255
Sport	16	65535
Dport	16	65535
Seq	32	4294967295
Ack	32	4294967295
Tcpflags	8	255
Win	16	65535
Carácter de contenido	8	255

Tabla 1. Campos a normalizar para su posterior análisis

4. ESTRUCTURA DE LA RED.

Una vez seleccionadas las entradas a la red (9 encabezados y 393 caracteres de contenido) el número de neuronas a las entradas que debe tener esta es de 402. La cuestión en este momento era ¿cuantas salidas se necesitan?.

Inicialmente, de la muestra obtenida de paquetes peligrosos (333 paquetes) SNORT los clasificó en 23 categorías o tipos de ataques, cada tipo de ataque se codificó en binario (00000001 = 1 … 00010111 = 23), de esta manera, con 8 bits se podría identificar hasta 255 tipos de ataques diferentes. Así la estructura de la red tendría 402 entradas, 8 salidas y un número por determinar de neuronas en la capa oculta.

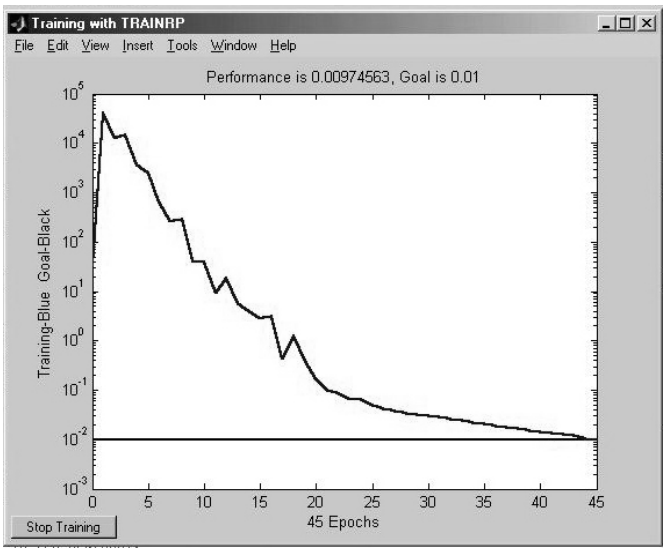
Se experimentó ampliamente con esta configuración, modificando el número de neuronas ocultas, y los algoritmos de entrenamiento, pero los resultados no fueron los esperados, los problemas de generalización eran evidentes.

Luego de verificar de nuevo los patrones de entrenamiento y la estructura de la red, se optó por una segunda aproximación anteriormente observada en algunos antecedente a este proyecto [1][3]. Se modificó la salida de la red a solo una neurona encargada de entregar dos resultados, 0 para patrones normales y 1 para patrones sospechosos o peligrosos.

Finalmente esta aproximación cumplió con las expectativas, generando así la estructura de 402 neuronas

de entrada, 410 neuronas en la capa oculta y 1 neurona de salida.

De todas maneras considerando que para el universo de datos elegido se había descartado una parte del contenido del paquete y como se podría presentar la posibilidad de que un ataque se encontrará en la porción descartada, se decidió hacer una prueba con todo el contenido para observar que tanto mejoraba la sensibilidad del sistema al tener todo la información sabiendo de antemano que el paquete maligno mas grande era 403 bytes (con cabeceras incluidas).




```
TRAINRP, Epoch 0/100000, MSE 33.1005/0.01, Gradient 279.984/1e-006
TRAINRP, Epoch 20/100000, MSE 0.169702/0.01, Gradient 6.3691/1e-006
TRAINRP, Epoch 40/100000, MSE 0.0145034/0.01, Gradient 0.1647/1e-006
TRAINRP, Epoch 45/100000, MSE 0.00974563/0.01, Gradient 0.0821558/1e-006
TRAINRP, Performance goal met.
```

Figura 1. Proceso de entrenamiento para trainrp con todo el contenido

Debido al número de neuronas, el proceso de entrenamiento tardo cerca de 3 horas, lo que comparado con los procesos de entrenamiento de la red elegida es demasiado lento.

Los resultados obtenidos fueron los siguientes:

Algoritmo utilizado	trainrp
Tolerancia	0,01
% éxito entrenados	98,80
% éxito paq. normales	83,78
% éxito paq. peligrosos	100

Tabla 2. Resultados obtenidos con todo el contenido del paquete

5. ENTRENAMIENTO DE LA RED.

Para el entrenamiento es necesario reunir en un único archivo una mezcla de los paquetes que se usarán para enseñar a la red neuronal, para este fin se usó una pequeña aplicación que usa como insumo 4 archivos, uno con los paquetes ya normalizados de tráfico normal y otro de paquetes considerados peligrosos, y otros dos con las respectivas salidas (un archivo con "ceros" para identificar el tráfico normal y otro con "unos" para identificar el tráfico peligroso). El resultado de esta etapa es un gran archivo que intercala cada uno de los tipos de paquetes. Dado que se han seleccionado 3 veces más paquetes normales que peligrosos, una vez que se han agotado los de menor cantidad, se ingresan de nuevo desde el primer paquete. Este proceso entrega un archivo que contiene los patrones de entrada y otro donde se encuentra la salida deseada para cada uno de los anteriores.

Los resultados obtenidos en las etapas anteriores fueron importados en MatLab, así los archivos planos separados por comas fueron cargados en el ambiente de trabajo en forma de matrices.

Los patrones de entrada, que están normalizados y sus 402 elementos ordenados se almacenaron en una matriz "p" de 402 elementos por 1998 paquetes (333 paquetes peligrosos y 999 paquetes normales, donde 999 es el numero mayor y la mezcla es uno a uno, el resultado obtenido son 1998 paquetes mezclados en total.). Las señales esperadas de salida (ceros o unos) se almacenaron en un vector "t" que permite corregir el error resultante de la red en la etapa de entrenamiento.

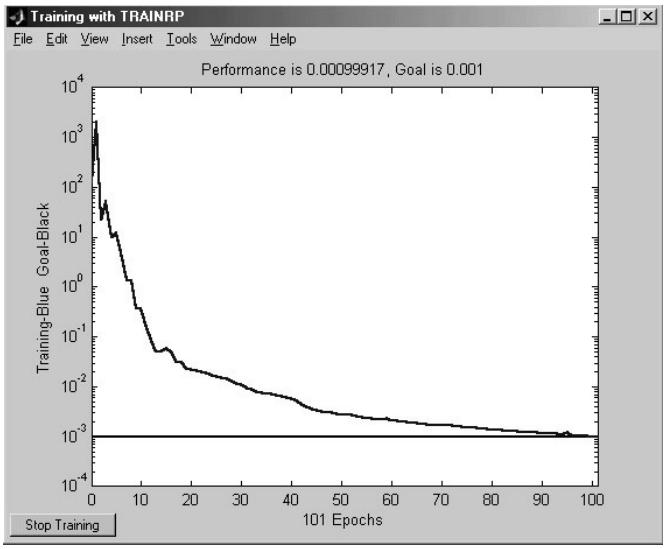
Una vez ejecutado el script de entrenamiento para alcanzar el objetivo (tolerancia máxima), se obtiene un resultado almacenado en el vector "salida_sim", este contiene valores entre 0 y 1 para cada patrón de entrada (matriz "p") que identifican si se ha reconocido el patrón como normal o peligroso. Si se resta esta salida (salida_sim) con la esperada (t) y se obtiene su valor absoluto se calcula una medida de dispersión para cada valor. Solo un patrón tuvo una dispersión mayor a 0.3 por lo tanto el porcentaje de efectividad estuvo alrededor de 99.94% para los patrones de entrenamiento.

Los entrenamientos más exitosos se obtuvieron con las siguientes funciones de aprendizaje:

Traincgb: Función de entrenamiento que actualiza los pesos y los bias de acuerdo con el algoritmo de retropropagación de gradiente conjugado según Powell-Beale.

Traincgp: Función de entrenamiento que actualiza los pesos y los bias de acuerdo con el algoritmo de retropropagación de gradiente conjugado según Polak-Ribiere.

Trainrp: Función de entrenamiento que actualiza los pesos y los bias de acuerdo con el algoritmo de "resilient backpropagation" (RPROP).




```
>> TRAINRP, Epoch 0/100000, MSE 121.407/0.001, Gradient 379.181/1e-006
TRAINRP, Epoch 20/100000, MSE 0.0223492/0.001, Gradient 0.253412/1e-006
TRAINRP, Epoch 40/100000, MSE 0.00576314/0.001, Gradient 0.0996112/1e-006
TRAINRP, Epoch 60/100000, MSE 0.00212861/0.001, Gradient 0.0520002/1e-006
TRAINRP, Epoch 80/100000, MSE 0.00136192/0.001, Gradient 0.0551182/1e-006
TRAINRP, Epoch 100/100000, MSE 0.00100915/0.001, Gradient 0.0167227/1e-006
TRAINRP, Epoch 101/100000, MSE 0.00099917/0.001, Gradient 0.0445956/1e-006
TRAINRP, Performance goal met.
```

Figura 2. Proceso de entrenamiento para trainrp con los contenidos seleccionados.

6. EL PROTOTIPO - CONSIDERACIONES

El estudio fue limitado para su viabilidad al análisis del tráfico de paquetes TCP únicamente, protocolos como UDP, ICMP, entre otros no serán tenidos en cuenta en esta aproximación.

Se usó Ethereal para obtener un archivo en formato TCPDUMP de los paquetes observados en la red de usuarios seleccionada, una sesión de captura se dedicó a registrar paquetes mientras el host utilizado era atacado usando la batería Nessus, dando como resultado un archivo de paquetes que contenían casi en su totalidad algún tipo de ataque. Otra sesión fue destinada a la obtención de tráfico normal mientras el host navegaba en Internet y realizaba otras actividades normales en red.

Para restringir el universo de ataques se inhabilitaron la mayoría de paquetes de reglas de Snort. Así los ataques se limitaron a los siguientes: *exploit.rules, web-php.rules, web-attacks.rules, backdoor.rules, shellcode.rules.*

El archivo obtenido de ataques fue filtrado entonces con Snort el cual identificó a 333 paquetes como ataques según las reglas activadas, de estos ataques el 80% fue utilizado para entrenamiento mientras el otro 20% sería usado como control una vez finalizado el entrenamiento de la red neuronal. Así mismo se seleccionaron del archivo de tráfico normal un número tres veces mayor que el de paquetes con ataques (999 paquetes) para el entrenamiento, y otros 333 fueron destinados para control.

Aproximaciones anteriores [1] habían considerado que el análisis de una cantidad considerable de información del contenido de cada paquete era sumamente difícil de analizar y por lo tanto era ignorada completamente, para el análisis del tráfico solo eran considerados algunos parámetros de los encabezados de cada protocolo.

En esta aproximación la información contenida en los paquetes es muy importante, por esta razón además de usar la información más significativa de los encabezados también se seleccionaron los primeros 393 caracteres del contenido del paquete. Este límite fue seleccionado dado que fue el número máximo de caracteres encontrado en nuestro universo de ataques detectados por Snort.

7. EL PROTOTIPO - FUNCIONALIDADES

El prototipo desarrollado cuenta con los siguientes pasos para realizar la detección:

Carga de parámetros. Los pesos obtenidos en la etapa de entrenamiento usando MatLab deben ser ingresados al prototipo para la identificación, estos pesos son exportados a archivos tipo CSV (separado por comas), estos archivos planos son cargados y almacenados en las respectivas matrices y vectores.

Cuatro archivos son importados pe.csv (pesos a la entrada), ps.csv (pesos a la salida), be.csv (bias a la entrada), bs.csv (bias a la salida).

Captura de Paquetes. Usando la librería pcap se captura un paquete a la vez, así sea directamente de la interfaz de red colocada en modo promiscuo o desde un archivo previamente almacenado en formato TCPDUMP.

Filtrado de Paquetes. Dados los limites de la investigación se ha decidido que solo será analizado el tráfico TCP, una vez capturado el paquete si cumple con la condición es analizado, de lo contrario el paquete es ignorado.

Normalización de Paquetes. En esta etapa se normaliza la información de encabezados y contenidos según lo tratado anteriormente. La información seleccionada es extraída del paquete y normalizada según la longitud en bits del campo en cuestión. Cada dato normalizado es almacenado en el vector data.

Análisis de Paquetes. El vector resultante de la normalización es usado en esta etapa para propagarlo hacia adelante a través de la red neuronal, este algoritmo

usa también los parámetros cargados en la etapa inicial, luego de realizar las operaciones matemáticas necesarias se obtiene como resultado la calificación para el paquete, este valor continuo se encuentra entre 0 y 1. Si es mas cercano a cero (0) el paquete es clasificado como normal y si es mas cercano a uno (1) es clasificado como peligroso. El margen de tolerancia se ha ubicado en 30% de la siguiente manera:

Menor a 0.3	Normal
Mayor a 0.31 y menor a 0.69	Incertidumbre
Mayor a 0.7	Peligroso

Tabla 3. Margen de tolerancia para la identificación de paquetes.

Reporte de Resultados. Una vez finalizado el análisis se obtiene métricas de desempeño como: Número de paquetes analizados, número de paquetes Normales, Peligrosos y en Incertidumbre.

8. RESULTADOS OBTENIDOS.

Métricas de Desempeño. Luego de realizar muchos entrenamientos con diferentes configuraciones de algoritmos y funciones de transferencia, se observó que los desempeños satisfactorios se encontraban especialmente con los siguientes cuatro algoritmos: Traincgb, Traincgp, Trainrp, Traincgf.

Algoritmo utilizado	Tolerancia	% éxito entrenados	% éxito paq. normales	% éxito paq. peligrosos
traincgb	0,01	98,09	89,78	94,11
	0,001	99,94	90,09	85,29
traincgp	0,01	97,84	87,98	97,05
	0,001	100	91,89	91,17
trainrp	0,01	99,80	96,09	94,11
	0,001	99,94	97,89	97,05
traincfg	0,01	98,09	87,68	82,35

Tabla 4. Comparación de desempeño de los algoritmos más exitosos.

Según se observa en la tabla 4, al disminuir el valor de la tolerancia, en general aumentó el porcentaje de éxito al entrenamiento y al identificar paquetes de tráfico normal, pero una considerable disminución al tratar de generalizar con los paquetes peligrosos. El algoritmo traincfg tuvo menor desempeño al identificar paquetes peligrosos. También se puede observar en la tabla de resultados, como el algoritmo trainrp mejoró su desempeño en todos los sentidos al disminuir la tolerancia.

Lo que ocurre es: En la tabla 5 se observa el número de falsos positivos (cuando un paquete inofensivo es clasificado como peligroso) y falsos negativos (cuando un paquete peligroso es clasificado como inofensivo). Se observa que el algoritmo trainrp tiene un mejor desempeño.

Algoritmo utilizado	Tolerancia	Falsos Negativos	Falsos Positivos
traincgb	0,01	1	2
	0,001	2	13
traincgp	0,01	1	14
	0,001	0	6
trainrp	0,01	0	4
	0,001	0	1
traincfg	0,01	3	9

Tabla 5. Falsos negativos y falso positivos para cada algoritmo.

9. CONCLUSIONES Y RECOMENDACIONES

Se logró una capacidad de generalización y de tolerancia a las mínimas variaciones en los paquetes, con la que no cuentan los sistemas basados en reglas.

Se estableció una posibilidad complementaria para la supervisión de redes informáticas, como es el análisis de los registros (logs) de tráfico mediante herramientas basadas en redes neuronales, capaces de detectar peligros no tomados en cuenta por los sistemas de seguridad convencionales.

Se logró obtener un modelo práctico y funcional como apoyo en la elusiva tarea de la detección de intrusos.

La aplicación de redes neuronales para la detección de intrusos en redes informáticas puede ser de gran apoyo para los sistemas de detección tradicionales gracias a su capacidad de generalización y aprendizaje.

Un sistema basado en redes neuronales permite detectar nuevos paquetes peligrosos a partir del conocimiento previamente adquirido, lo que no sucede con los sistemas no basados en redes neuronales. Esto permite que el administrador de una red se convierta en un contribuyente de conocimiento y no simplemente en un usuario de sistemas de detección de intrusos.

El prototipo desarrollado demostró que en este momento un sistema de detección de intrusos basado en redes neuronales sirve como herramienta de apoyo a los actuales sistemas de detección.

El éxito de un sistema de redes neuronales se basa en el entrenamiento y en la selección de un universo adecuado y familiar a la red donde se planea utilizar el sistema.

Los niveles de incertidumbre que ofrecen las redes neuronales permiten que se puedan detectar patrones o comportamientos que para otros sistemas basados en

reglas son descartados debido a las características discretas de estos.

Una extensión de este trabajo sería el desarrollo de un modelo neuronal posiblemente modular capaz de diferenciar y clasificar los diferentes tipos de ataques que pueden aparecer en una red informática.

La creación de un sistema de entrenamiento daría la posibilidad de actualizar el sistema autónomamente, según las necesidades del administrador.

Se obtuvo un modelo viable para la detección de intrusos y el análisis de redes basado en redes neuronales.

9. BIBLIOGRAFÍA

[1] CORTADA, P; Sanromá, G.; García, P; Arenas A. y Rallo R.. IDS basado en Mapas Autoorganizados. Salamanca, noviembre de 2002

[2] GREDIAGA, Angel; Ibarra, Francisco; Ledesma, Bernardo; Brotons, Francisco. Aplicación de redes neuronales en IDS y sus ventajas. Utilización de redes neuronales para la detección de intrusos. [en línea], Febrero 2002. [citado en 10 de septiembre 2003]. Disponible en Internet: http://www.criptored.upm.es/guiateoria/gt_m093a.htm

[3] HILERA, Jose R. Redes Neuronales Artificiales: Fundamentos, Modelos y Aplicaciones. México : Alfaomega grupo editor, 2000. P390 ISBM 9586821722

[4] DÍAZ Vizcano, Luis Miguel. Sistema de Detección de Intrusos. Universidad Carlos III de Madrid. Departamento de Ingeniería Telemática. 2002