SISTEMA INMUNOLÓGICO PARA LA DETECCIÓN DE INTRUSOS A NIVEL DE PROTOCOLO HTTP

EFRAIN TORRES MEJIA

Proyecto de grado presentado para optar al título de Ingeniero de Sistemas

Director: Enrique Ruiz

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERIA
CARRERA DE INGENIERIA DE SISTEMAS
SANTAFÉ DE BOGOTA D.C.
MAYO DEL 2003

PONTIFICIA UNIVERSIDAD JAVERIANA

FACULTAD DE INGENIERIA

CARRERA DE INGENIERIA DE SISTEMAS

Rector Magnífico:
Padre Gerardo Remolina Vargas S.J.

Decano Académico Facultad de Ingeniería:
Ingeniero Roberto Enrique Montoya Villa

Decano del Medio Universitario Facultad de Ingeniería:
Padre Álvaro González Sánchez S.J.

Director Carrera de Ingeniería de Sistemas:
Ingeniero Javier Francisco López Parra

Director Departamento de Ingeniería de Sistemas:
Ingeniero José Hernando Hurtado Rojas

Nota de Aceptación

_____

_____

_____

_____

Director del proyecto

_____

Jurado

_____

Jurado

Junio 2003

Artículo 23 de la Resolución No. 1 de Junio de 1946

“La Universidad no se hace responsable de los conceptos emitidos por sus alumnos en sus proyectos de grado.

Sólo velará porque no se publique nada contrario al dogma y la moral católica y porque no contengan ataques o polémicas puramente personales. Antes bien, que se vean en ellos el anhelo de buscar la verdad y la Justicia.”

TABLA DE CONTENIDO

TABLA DE FIGURAS

INTRODUCCIÓN

"Uno tiene que acercarse a lo que es radicalmente nuevo con una mente en blanco, rechazando conscientemente el intento de ligar la historia con lo que ya es familiar, porque lo familiar es desesperadamente inadecuado."

(On the cruelty of really teaching computing science, 1989 - Edsger W. Dijkstra)

En los últimos años la generación de nuevas tecnologías para la detección de intrusos solo se ha presentado en el ámbito académico, sin dar el salto final necesario. Pasar a la realidad. Solo un pequeño acercamiento a este objetivo es el objetivo principal de este trabajo. Retomar las innumerables investigaciones realizadas y relegadas sobre el tema para replantearlas, adecuarlas y transformar sus enfoques hacia la realidad de la seguridad informática. Este trabajo también pretende eliminar las limitaciones de los sistemas de detección actuales, siguiendo una metodología que ha demostrado ser siempre efectiva en el vasto campo de la seguridad informática, "Keep It Simple!" o vulgarmente conocida como K.I.S.S.

Los sistemas comerciales actuales además de ser sumamente costosos y complejos, teniendo en cuenta su labor, siguen estancados en modelos que no se han modificado, lo cual los hace candidatos a quedar relegados, si acaso ya no lo estan. Relegados por el desarrollo de técnicas anti-ids, técnicas que desde hace mucho tiempo existían y que hasta los mismos académicos habían vislumbrado en sus desarrollos. Quizás el salto a la realidad no se ha dado debido a la educación teórica, encasillada y poco práctica de la seguridad informática en ambientes universitarios. Este trabajo surge de las inquietudes e ideas generadas en los últimos 6 años, esperando que sirva de ayuda a las

nuevas generaciones de personas que creen en la libertad de la información, que la curiosidad responsable no es un crimen y que querer es poder.

El siguiente trabajo fue evaluado por jurados internacionales y aceptado para su presentación en las III Jornadas Nacionales de Seguridad Informática - ACIS 2003, evento realizado por la Asociación Colombiana de Ingenieros de Sistemas (ACIS) http://www.acis.org.co.

1. OBJETIVOS

Los objetivos listados corresponden a los objetivos planteados en la propuesta definitiva de proyecto de grado [0].

OBJETIVO GENERAL

El objetivo general de este proyecto es el diseño, desarrollo e implementación de un sistema de detección de intrusos a nivel de red (NIDS) para protocolo HTTP, utilizando nuevas tecnologías adaptivas, desarrolladas para el diseño de HIDS.

OBJETIVOS ESPECIFICOS

- Implementar un sistema prototipo NIDS para la detección de ataques a nivel de protocolo HTTP.

- Introducir una nueva infraestructura básica para la detección de intrusos dirigido al nivel de aplicación que reduzca las limitaciones de los IDS actuales.

- Establecer la viabilidad de los sistemas inmunológicos computacionales multinivel. Haciendo énfasis en la especialización de los niveles de detección.

- Recopilar y analizar información para determinar el nivel adecuado de abstracción de los datos de entrada para las redes neuronales en la detección de intrusos.

- Determinar la viabilidad de las redes Elman para su utilización en la detección de intrusos a nivel del protocolo HTTP.

2. ALCANCE

El alcance de este proyecto esta definido por los objetivos generales y específicos aprobados en los que se basa la realización misma de este trabajo. El proyecto esta enfocado al desarrollo de un prototipo IDS funcional que tienda a eliminar las limitaciones de los IDS actuales, replanteando ciertos enfoques actuales, la mayoría de ellos provenientes del desarrollo académico de HIDS. Aunque este proyecto tiene un alcance claro, es campo fértil para la continuación de nuevas investigaciones que permitan generar nuevas tecnologías aplicables a la realidad de la seguridad informática.

El desarrollo mismo del prototipo permitirá replantear ciertas consideraciones a nivel de la seguridad de las aplicaciones en el protocolo HTTP, teniendo en cuenta que la Word Wide Web es el servicio público de Internet por excelencia.

4. JUSTIFICACIÓN

Es notable el incremento en la investigación sobre nuevas tecnologías para el desarrollo de sistemas de detección de intrusos. Muchos de estos estudios presentan enfoques radicales que pretenden mostrar soluciones globales a nivel de detección de ataques y por ende la prevención de estos. Desafortunadamente la implementación de estas grandes soluciones en un ambiente real ha sido casi nula. Mientras tanto las organizaciones siguen adquiriendo los mismos sistemas comerciales de alto costo para la detección de intrusos, los cuales han probado ser ineficientes y poco efectivos ante nuevas técnicas de ataque.

5. ANTECEDENTES

5.1 SISTEMAS DE DETECCIÓN DE INTRUSOS

Es muy importante que los mecanismos de seguridad de un sistema estén diseñados e implementados de tal forma, que prevengan el acceso no autorizado a sus recursos y datos. Sin embargo prevenir los problemas de seguridad en su totalidad, es una tarea imposible debido al factor humano. Pero es posible seguir minimizando los riesgos, tratando de detectar los intentos de ataques o intrusiones para poder tomar las acciones necesarias y reducir su impacto. En 1980, mientras introducía el concepto de detección de intrusos, Anderson [45] definió una intrusión como la posibilidad potencial de un intento deliberado, no autorizado, de acceder información, manipularla o eliminar la funcionalidad del sistema.

A los sistemas utilizados para detectar las intrusiones o los intentos de intrusión se les denomina sistemas de detección de intrusiones (Intrusion Detection Systems, IDS) o normalmente, sistemas de detección de intrusos. Cualquier mecanismo de seguridad con este propósito puede ser considerado un IDS, pero generalmente sólo se aplica esta denominación a los sistemas automáticos y especializados en desarrollar este tipo de tareas de detección.

Generalmente existen dos grandes enfoques a la hora de clasificar a los sistemas de detección de intrusos: en función de qué sistemas vigilan, o en función de cómo lo hacen. Si tomamos la primera de estas aproximaciones los sistemas de detección de intrusos generalmente están divididos en 2 clases, los sistemas de detección de intrusos a nivel de red o NIDS y a nivel de host o HIDS. Existe una nueva tercera clase la cual ha sido denominada sistema de

detección de intrusos a nivel de aplicación o ApplicationIDS, esta ultima clase también ha sido denominada como "Firewall de aplicación".

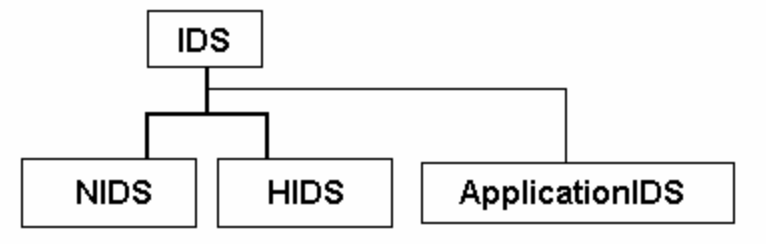

Figura 1. Clasificación de los IDS

Desafortunadamente, hoy en día, estos sistemas basan su funcionamiento en el reconocimiento de firmas de ataques previamente conocidos, las cuales son comparadas, en el caso de los NIDS contra el tráfico de red capturado por un sensor previamente establecido, o en el caso de los ApplicationIDS contra el tráfico de red dirigido hacia un servidor determinado, el cual generalmente es un servidor Web.

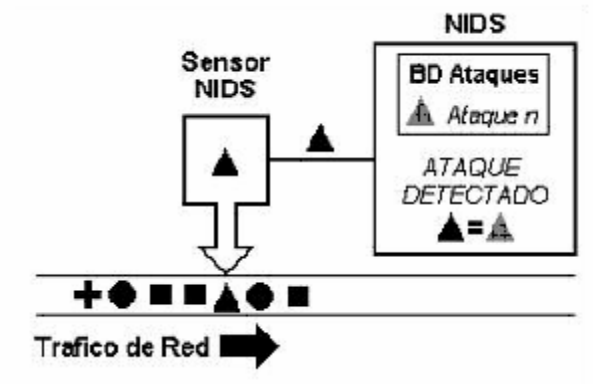

Figura 2. Funcionamiento básico de un NIDS

Los HIDS han basado su funcionamiento en módulos que son aplicados directamente al kernel de un sistema en particular, añadiendo nuevas características de auditoria y seguridad que evitan que un usuario local escale privilegios, teniendo acceso a recursos y procesos de forma no autorizada. Además, se han desarrollado nuevas investigaciones enfocadas al desarrollo de tecnologías adaptivas basadas en sistemas inmunológicos para la detección de intrusos a nivel de host (HIDS).

En la actualidad esta división queda algo pobre, ya que cada día se avanza más en el diseño de sistemas de detección de intrusos que no tendrían cabida en ninguna de las categorías anteriores o que corresponderían a múltiples clases al mismo tiempo.

La segunda gran clasificación de los IDS se realiza en función de cómo actúan estos sistemas; actualmente existen dos grandes técnicas de detección de intrusos, las basadas en la detección de anomalías (anomaly detection) y las basadas en la detección de usos indebidos del sistema (misuse detection). En la detección de anomalías se supone que una intrusión se puede ver como una anomalía en el sistema, por lo que si se establece un perfil del comportamiento normal de los sistemas es posible detectar las intrusiones, una intrusión sería básicamente una desviación del comportamiento normal establecido. Para la detección de usos indebidos se establecen firmas para cada uno de los diferentes ataques conocidos y algunas de sus variaciones. Así, mientras que la detección de anomalías conoce lo normal y detecta lo que no lo es, la detección de usos indebidos se limita a conocer lo anormal para poderlo detectar, es esta última técnica la que comúnmente se utiliza en los IDS actuales, la cual posee grandes limitaciones.

5.1.1 IDS Actuales

En la actualidad existe una gran cantidad de productos, tanto de libre distribución como comerciales, que cumplen las necesidades especificas de una determinada infraestructura, ya sea a nivel de rendimiento, nivel de configuración o mantenimiento, aunque en su funcionamiento interno compartan básicamente la misma arquitectura y funcionamiento según su tipo. Prácticamente todos los sistemas de detección de intrusos tienen las siguientes partes bien definidas:

- **Recolectores de datos o Sensores:** Programas que capturan los datos de la fuente para su análisis en tiempo real o posterior.

- **Reglas o firmas:** Estas reglas son las que caracterizan las violaciones que pueden ser cometidas y contra las que se contrastan los datos obtenidos en el punto anterior.

- **Filtro:** Esta parte se encarga de contrastar las reglas o firmas contra los datos obtenidos.

- **Generador de informes o alarmas:** Después de procesar los datos contra las reglas por el filtro, si existe alguna situación que haga creer que se ha vulnerado o intentado vulnerar la seguridad del sistema, esta parte del detector de intrusos informa este hecho.

- **Mecanismo de administración:** Para facilitar la administración del IDS y monitorear su estado, se desarrollan interfazes gráficas amigables las cuales permiten a los administradores, realizar el mantenimiento y control del IDS.

El siguiente es un listado de los IDS más conocidos y representativos, actualmente en el mercado:

Snort: http://www.snort.org

Este IDS es el más famoso de libre distribución de los cientos que hay disponibles en el mercado. Está basado en análisis de patrones. Recibe múltiples colaboraciones que permiten mantener su base de datos de firmas muy actualizadas. Además, constantemente hay gente programando nuevos plugins y programas externos que le añaden nuevas funcionalidades, lo que permite un alto grado de personalización y estar al día en nuevas características, lo cual hace que snort sea copiado en parte por otros productos comerciales y su uso sea extensivo en ambientes corporativos y gubernamentales [46]. Existe una versión para Windows2000/NT. Por su nivel tecnológico, sus características y su reputación en el mercado, se convierte en un digno IDS para ser comparado con otros.

RealSecure http://www.iss.net

RealSecure es un sistema de detección de intrusos comercial, el cual esta dividido en dos partes: directores, que se utilizan para tareas administrativas y operativas, y sensores, que son los generadores de eventos. Están disponibles los sensores basados en red y basados en host. Tienen versiones para UNIX y para Windows NT/2000, pero la consola sólo se ejecuta en NT/2000. RealSecure utiliza normativas para definir eventos de interés. Estas normativas se configuran en el director de análisis y se descargan desde los sensores. El sensor la toma y la utiliza en la detección de eventos. Si la consola está funcionando los eventos se toman en tiempo real.

NetRanger http://www.cisco.com

Cisco lo llama “componente de seguridad dinámico”. Se puede configurar el sistema para detectar, informar y responder automáticamente a algunos eventos

de interés. Las capacidades de alerta pueden ser beeps, ventanas desplegadas, alertas de busca o e-mails. La respuesta automática incluye la capacidad de reiniciar conexiones o reconfigurar enrutadores. Los componentes del sistema incluyen sensores y una estación de análisis Estos componentes se comunican mediante un protocolo patentado.

Dragon: http://www.enterasys.com/ids/

Dragon está compuesto por tres productos, Dragon Sensor, Dragon Squire y Dragon Server. Cada uno de los tres productos que configuran Dragon 5 presenta unas funciones diferentes. Por un lado, Dragon Sensor es un sistema de detección de intrusiones basado en la red de alta velocidad, mientras que Dragon Squire se emplea para detener los ataques que se producen en los hosts locales y Dragon Server agrupa todas las características del producto mediante la gestión y generación de informes centralizados.

NFR: http://www.nfr.com/

El Network Flight Recorder es un IDS que consta básicamente de un nucleo que captura todo el tráfico de la red y lo pasa a los denominados "packages" para que lo procesen y tomen acciones si fuera necesario. Estos “packages” son generados en un sencillo lenguaje denominado N-Code. El manejo de todo el conjunto se realiza a traves de una interfaz java accesible con un navegador web.

Para una lista más detallada se recomienda el listado de IDS de Talisker en http://www.networkintrusion.co.uk/, la cual hasta el momento posee mas de 100 referencias actualizadas.

5.2 SISTEMAS INMUNOLÓGICOS COMPUTACIONALES

El sistema inmunológico humano es un gran ejemplo de sistema para la detección de intrusos, el cual elimina muchos de los problemas anteriormente evidenciados. Este sistema es un sistema distribuido, robusto, dinámico, diverso y adaptable. Lo cual lo convierte en una excelente base para la protección de sistemas computacionales [1][2]. Adicionalmente es muy importante recordar que el sistema inmunológico es un sistema de protección por niveles donde el agente patógeno se enfrenta principalmente a la piel como barrera inicial y posteriormente al sistema como tal, el cual basa su funcionamiento en el reconocimiento de patrones por parte de los linfocitos. Dicho reconocimiento no se basa en una memoria estática y permanente, es una memoria evolutiva, heredada y por lo tanto dinámica.

Existen estudios donde se han utilizado estos conceptos inmunológicos para crear un modelo de detección de intrusos [13] [14], reconociendo dinámicamente patrones de ataques en los llamados al sistema de procesos privilegiados [3][4]. Dichos estudios mostraron que es factible utilizar el reconocimiento de patrones aprendidos previamente en los llamados al sistema para detectar agentes patógenos que representan intrusiones al sistema.

Desafortunadamente al manejar la información a ese nivel se pierde información muy valiosa y específica que se encuentra en niveles superiores de abstracción. El nivel óptimo para analizar dicha información corresponde a la del mismo protocolo de aplicación utilizado. Es muy distinto analizar un comportamiento de el protocolo SMTP en comparación al de una sesión en el protocolo HTTP, al llegar cualquier requerimiento al host destino se corre el riesgo que al no ser detectado como comportamiento anómalo (intrusivo), ya el proceso infectó el sistema. Lo mismo ocurre con los sistemas actualmente desarrollados en proyectos de investigación en el análisis de comportamientos, en los HIDS

actuales y en los ApplicationIDS pero estos últimos generalmente manejan un tipo de barrera especial la cual define unas características válidas para un requerimiento determinado, si logra pasar dicho filtro, el cual es a nivel de aplicación, sigue el proceso de identificación. En el enfoque inmunológico esta barrera inicial serviría de piel para el sistema. Si el requerimiento logra superar esta barrera se enfrentaría al sistema inmunológico, el cual por medio de la detección de patrones normales, aprendidos previamente, verificaría si dicho requerimiento corresponde a un comportamiento normal y por lo tanto puede ser procesado. De lo contrario el requerimiento es desechado y puede opcionalmente pasar a una etapa adicional de identificación.

Para evitar los problemas con las limitaciones presentadas con respecto a los HIDS e ApplicationIDS, el sistema propuesto debe ser genérico, para lo cual es posible desarrollar un Proxy externo a la aplicación a proteger. Así no importaría que tipo de aplicación se tiene, adicionalmente, puede ser distribuido para que en el caso de ser vulnerado no perjudique al servicio que protegía. Este enfoque a nivel de arquitectura hace frente a las propias limitaciones que los ejemplos biológicos reales poseen. Así aunque muchos estudios sobre algoritmos genéticos y redes neuronales artificiales recalquen que la evolución de dichas tecnologías se basa en la copia de características, pero no en la replicación computacional de un sistema biológico, en la práctica es difícil delimitar hasta donde se copia la naturaleza.

5.3 WORLD WIDE WEB (WWW) Y HTTP

Una organización típica con presencia en Internet, “protegida” por un firewall, permite tráfico HTTP o HTTPS hacia sus servidores Web. WWW se ha convertido en un servicio público por excelencia. En diciembre del 2002, la empresa de seguridad informática española S21SEC realizó un estudio durante los últimos cinco meses (junio-noviembre). En dicho estudio se pudo establecer

que de 2113 vulnerabilidades publicadas, 1320 vulnerabilidades tienen su origen en las aplicaciones Web [5]. Lo cual representa un 62,5% de las vulnerabilidades reportadas, dicho porcentaje refleja no solo la gran cantidad de problemas de seguridad en las aplicaciones que soportan o manejan este tipo de servicio, si no también el alto riesgo al que están expuestas las organizaciones con una presencia Web hacia Internet. Riesgo que no ha sido realmente evaluado, teniendo en cuenta que la gran mayoría de organizaciones no conocen las limitaciones de mecanismos de protección tales como los firewalls. El panorama es mucho mas crítico si se tiene en cuenta la complejidad creciente de los servidores Web, lo cual gracias a la gran cantidad de vulnerabilidades en ellos y en las aplicaciones que soportan se han convertido en puntos de acceso no autorizados hacia cualquier organización. Debido a esto el sistema desarrollado esta dirigido directamente hacia el protocolo HTTP.

En la actualidad el protocolo HTTP se encuentra en la versión 1.1. HTTP/1.0 especifica que una sola conexión TCP provee un solo mensaje HTTP. HTTP/1.1 permite múltiples mensajes sobre una única conexión TCP [6]. Esta diferencia no afecta la detección de intrusos a nivel de aplicación, para lo cual se maneja la última versión como referencia sobre este protocolo.

5.4 LIMITACIONES DE LOS IDS ACTUALES

Los IDS actuales basan su funcionamiento en un esquema estático lo cual los convierte en sistemas muy ineficientes en el momento en el que un atacante codifica los datos de entrada. Adicionalmente, la base de datos de las firmas de ataques sufren los mismos inconvenientes que los antivirus, si la base de datos está desactualizada es muy probable que el sistema sea atacado sin ser detectado. Además, el tiempo entre el descubrimiento de un nuevo ataque y su

inclusión es bastante alto, teniendo en cuenta que el tiempo entre la publicación de dicho ataque y el uso indebido por parte de un posible atacante es muy corto.

Con respecto a los NIDS existen inconvenientes adicionales. Uno de ellos es la problemática generada por no poder capturar y analizar todo el tráfico que debe ser observado. Esta limitación proviene de los mismos analizadores de tráfico (sniffers), los cuales son los sensores del sistema. El sistema tiene que manejar problemas de red como la fragmentación de datos lo cual implica utilizar poder de procesamiento en la manipulación de datos, bajando el rendimiento del sistema y la probabilidad que detecte un ataque.

Los ApplicationIDS generalmente funcionan cubriendo la aplicación a proteger, lo cual los hace dependientes de la aplicación y del mismo software. Así, por ejemplo existen ApplicationIDS que solo son para Microsoft Internet Information Server (IIS) o Apache. Esto es una gran limitación en su implementación y puede generar nuevos problemas de seguridad ya que aumentan el nivel de complejidad de la aplicación a proteger, aumentando así la posibilidad de que se generen nuevos problemas de seguridad.

5.5 ANÁLISIS DE COMPORTAMIENTOS

Viendo las limitaciones de los IDS actuales, algunos centros de investigación en el mundo empezaron a analizar el comportamiento de los usuarios para determinar la existencia de anomalías para la identificación de ataques. Dicho análisis se ha enfocado básicamente a determinar los comportamientos anómalos, pero no los comportamientos normales. Este enfoque inicial implica la necesidad de conocer de antemano todos los comportamientos anómalos posibles, lo cual es mucho mas complejo que el determinar el comportamiento normal y así en el momento de analizar un comportamiento si no corresponde al

normal debe ser rechazado. Esto abre las puertas a falsos-positivos, lo cual es mas seguro que el tener falsos-negativos en un servicio critico determinado. Sin embargo la generación de falsos-negativos por parte de un IDS tiene repercusiones a nivel de pérdida de imagen por parte de los usuarios que acceden a un servicio determinado.

Los inicios del análisis de comportamientos se enfocaron principalmente en el desarrollo de mecanismos estadísticos, para determinar anomalías en el comportamiento de los usuarios de un sistema determinado. La gran ventaja que presenta un mecanismo estadístico es que este es fácilmente adaptable a las nuevas condiciones cambiantes en el tiempo. Pero esa adaptabilidad es susceptible a cambios progresivos programados, lo cual permite la inclusión de actividades intrusivas evitando su detección [7].

Generalmente existen dos tipos de análisis de comportamientos, el análisis de comportamiento para la detección de anomalías (anomaly detection) y de uso erróneo (misuse detection). La detección de anomalías puede ser definida como el intento de detectar intrusiones descubriendo desviaciones significantes del comportamiento normal. La detección de uso erróneo o indebido corresponde al enfoque actual donde se utilizan firmas de ataques previamente introducidas al sistema detector, las cuales son comparadas con actividades en el sistema para detectar un ataque.[7]

En 1999 el DARPA apoyo el estudio "Learning Program Behavior Profiles for Intrusion Detection " [7], se demostró la necesidad de analizar el comportamiento normal de los programas, es decir anomaly detection, vs. misuse detection, en pocas palabras debido a que misuse detection solo permite detectar ataques previamente conocidos, como se menciona en los antecedentes. En este estudio se utilizaron básicamente 3 algoritmos, de los cuales el mas eficiente demostró ser el algoritmo basado en redes neuronales artificiales Elman , las cuales poseen las características de las redes neuronales y por lo tanto la habilidad de

aprender y generalizar [7]. Diferentes estudios [7] [8] [9] han demostrado la viabilidad y efectividad del uso de redes neuronales artificiales para la detección de intrusos.

5.6 ENFOQUES INVESTIGATIVOS EN EL USO DE REDES NEURONALES ARTIFICIALES PARA LA DETECCIÓN DE INTRUSOS

La mayoría de los enfoques investigativos con respecto al uso de redes neuronales para la detección de intrusos han sido dirigidos hacia el desarrollo de HIDS. Los cuales basan su funcionamiento en la lectura y posterior análisis de llamados al sistema que han sido capturados por mecanismos de auditoria detallados como el BSM (Basic Security Module - Sun Microsystem) en ambientes operativos Solaris (Sun Microsystems). Este enfoque posee limitaciones muy graves, reflejadas también en los sistemas inmunológicos computacionales en desarrollo.

Es común confundir el proceso de detección con el de identificación de ataques [9]. Es importante anotar que al confundir estos dos procesos, el sistema desarrollado puede perder su enfoque inicial y por lo tanto su efectividad. En algunos casos es necesario para el desarrollo de un sistema de detección de intrusos, conocer de antemano una clasificación o taxonomía de ataques la cual, además de ser compleja, implica la capacidad de ubicar ataques conocidos y desconocidos dentro de una clasificación estática. Esto es opuesto a la detección de ataques desconocidos. Otro error común es el tratar de adicionar características tales como la detección de ataques exitosos [9]. Para lo cual hay que tener en cuenta que un ataque solo es exitoso si no es detectado, asumiendo que el ataque fue desarrollado en el ambiente propicio para su efectivo funcionamiento. La única viabilidad de tener un sistema de detección de

intrusos con dichas características correspondería a la puesta en funcionamiento de un Honeypot.

6. MARCO TEÓRICO

6.1 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 que a diferencia del las células del resto del cuerpo no se regeneran. Porque este tipo de célula es la única parte del cuerpo que no se regenera lentamente, se asume que estas células son las que nos proveen las capacidades de recordar, pensar, y aplicar experiencias anteriores a cada una de nuestras acciones. Estas células, 100 billones de ellas aproximadamente, se conocen como neuronas. Cada una de estas neuronas puede conectarse hasta con otras 200.000 neuronas, aunque 1.000 a 10.000 conexiones es lo típico.

La potencia de la mente humana viene del funcionamiento poco conocido de estos componentes básicos y de las múltiples conexiones entre ellas. También viene de la programación genética y del aprendizaje.

Individualmente las neuronas son complicadas. Tienen una variedad de piezas, subsistemas, y de mecanismos de control. Transportan la información por medio de caminos electroquímicos. Hay más de 100 clases diferentes de neuronas, dependiendo del método de la clasificación usado. Juntas, estas neuronas y sus conexiones forman un proceso que no es binario, no es estable, y no sincrónico. En pocas palabras no es nada como los computadores electrónicos actualmente disponibles, o aún, las redes neuronales artificiales.

Estas redes neuronales artificiales solo intentan replicar los elementos más básicos de este complicado, versátil, y poderoso organismo. Lo hacen de una manera primitiva. Pero para el programador que está intentando solucionar un problema determinado, la computación neuronal nunca fue sobre replicar cerebros humanos. Es sobre máquinas y sobre una nueva manera de solucionar problemas.

Las redes neuronales artificiales son básicamente crudos modelos electrónicos basados en la estructura neuronal del cerebro. El cerebro aprende básicamente por experiencia.

Estos métodos biológicamente inspirados de computar se piensan como el siguiente gran adelanto en la industria de la computación. 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.

Ahora, los avances en la investigación biológica prometen una comprensión inicial del mecanismo natural del pensamiento. Esta investigación muestra que los cerebros salvan la información como modelos. Algunos de estos modelos son muy complicados y nos permiten reconocer caras individuales desde diversos ángulos. Este proceso de salvar la información como modelos, de utilizar estos modelos, y después solucionar problemas abarca un nuevo campo en la computación. Este campo, según lo mencionado antes, no 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 [10].

6.2 NEURONAS ARTIFICIALES

El elemento fundamental de procesamiento en una red neuronal es una neurona. Esta unidad del conocimiento humano abarca algunas capacidades generales. Básicamente, una neurona biológica recibe entradas de información de otras fuentes, las combina de una cierta manera, realiza una operación generalmente no lineal en el resultado, y después hace salir el resultado final.

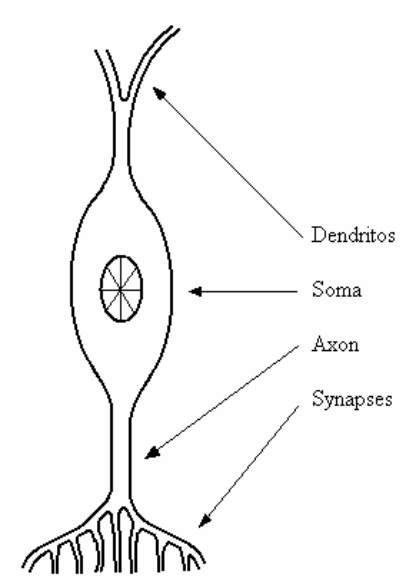

Figura 3. Las 4 partes básicas de la neurona

Los dendritos son los encargados de recibir las entradas, el soma procesa las entradas, el axon convierte las entradas procesadas en salidas y los synapses son el contacto electromecánico con otras neuronas.

Nuevos datos experimentales recientes han proporcionado evidencia adicional para determinar que las neuronas biológicas son estructuralmente más complejas que la explicación anterior. Son considerablemente más complejas que las neuronas artificiales existentes. Mientras que la biología proporciona una

comprensión mejor sobre las neuronas, y mientras que la tecnología avanza, los diseñadores de redes neuronales pueden continuar mejorando sus sistemas mediante la comprensión biológica del cerebro humano. Actualmente, la meta de las redes neuronales artificiales no es la reconstrucción del cerebro. Al contrario los investigadores están buscando una mayor comprensión de las capacidades de la naturaleza por las cuales la gente puede ingeniar soluciones a problemas que no han sido solucionados por medio de la computación tradicional [10].

Para hacer esto, la unidad básica de las redes neuronales artificiales, la neurona artificial, simula las cuatro funciones básicas de las neuronas naturales.

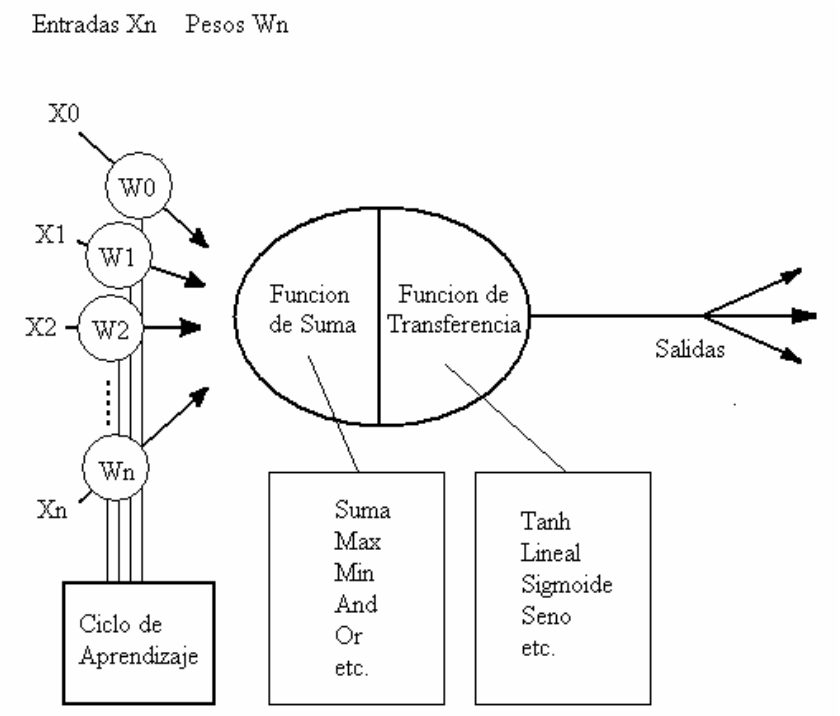

Figura 4. Modelo de una neurona artificial

Varias entradas de información a la red son representadas por el símbolo matemático, (Xn). Cada una de estas entradas de información es multiplicada

por un peso de la conexión (Wn). En el caso más simple, estos productos se suman simplemente alimentado a la función de transferencia. Esta función entonces da vuelta a este número en una salida verdadera por medio de un cierto algoritmo para generar un resultado, y así una salida. La cual se convierte en la entrada a otras neuronas artificiales conectadas a esta.

La otra parte con respecto al uso de redes neuronales, corresponde a la forma como estas neuronas individuales pueden ser agrupadas. Básicamente todas las redes neuronales artificiales tienen una estructura similar o topología a la siguiente:

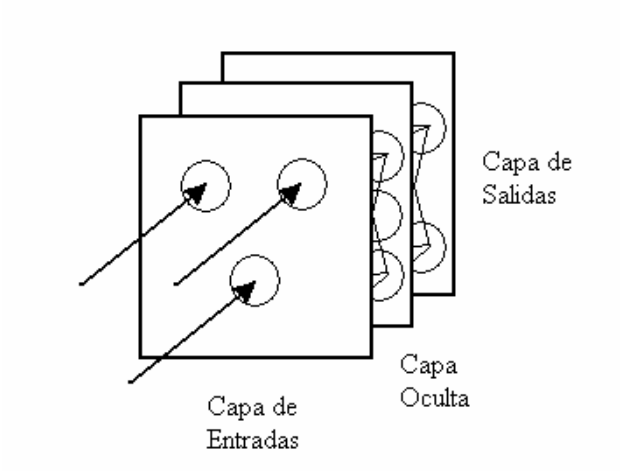

Figura 5. Topología por capas de una red neuronal

En esta estructura algunas neuronas son interfaces con el mundo real para recibir las entradas. Otras neuronas proveen al mundo real con las salidas de la red. El resto de las neuronas están en capas intermedias, ocultas. Una red neuronal es más que un conjunto de neuronas. En la mayoría de las redes, cada neurona en una capa oculta recibe las señales provenientes de todas las neuronas que se encuentran en una capa superior, típicamente una capa de entradas. Estas líneas de comunicación de una neurona a otra son aspectos

importantes de las redes neuronales. Estas son las conexiones que proveen la fuerza variable que excita o inhibe a otras neuronas.

6.3 ENTRENAMIENTO DE UNA RED NEURONAL ARTIFICIAL

Una vez que la red haya sido estructurada para una aplicación en particular, la red esta lista para ser entrenada. Para comenzar este proceso los pesos iniciales son escogidos aleatoriamente. Así el entrenamiento o aprendizaje comienza.

Existen dos tipos de enfoques con respecto al entrenamiento, entrenamiento supervisado y no supervisado. El entrenamiento supervisado involucra un mecanismo que provee a la red con la salida deseada, dicho mecanismo puede ser la calificación manual de la efectividad de la red o proveer las salidas deseadas con las entradas. Así, la red procesa las entradas, compara las salidas con las salidas deseadas y propaga los errores en las capas anteriores, ajustando los pesos que controlan la red. Este proceso se repite una y otra vez hasta que se halla alcanzado el estado deseado. El entrenamiento no supervisado es cuando la red tiene que dar sentido a las entradas sin ayuda externa. La gran mayoría de redes neuronales artificiales utilizan entrenamiento supervisado ya que el otro enfoque todavía no es totalmente comprendido, y por lo tanto generalmente no funciona bien, así que sigue estando relegado a los laboratorios de investigación [10].

6.4 RED NEURONAL ELMAN

En 1990, el Doctor Jeffrey L. Elman en su trabajo "Finding Structure in Time" [11] desarrolló una red neuronal basada en el trabajo previo de Jordan [47] (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, 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.

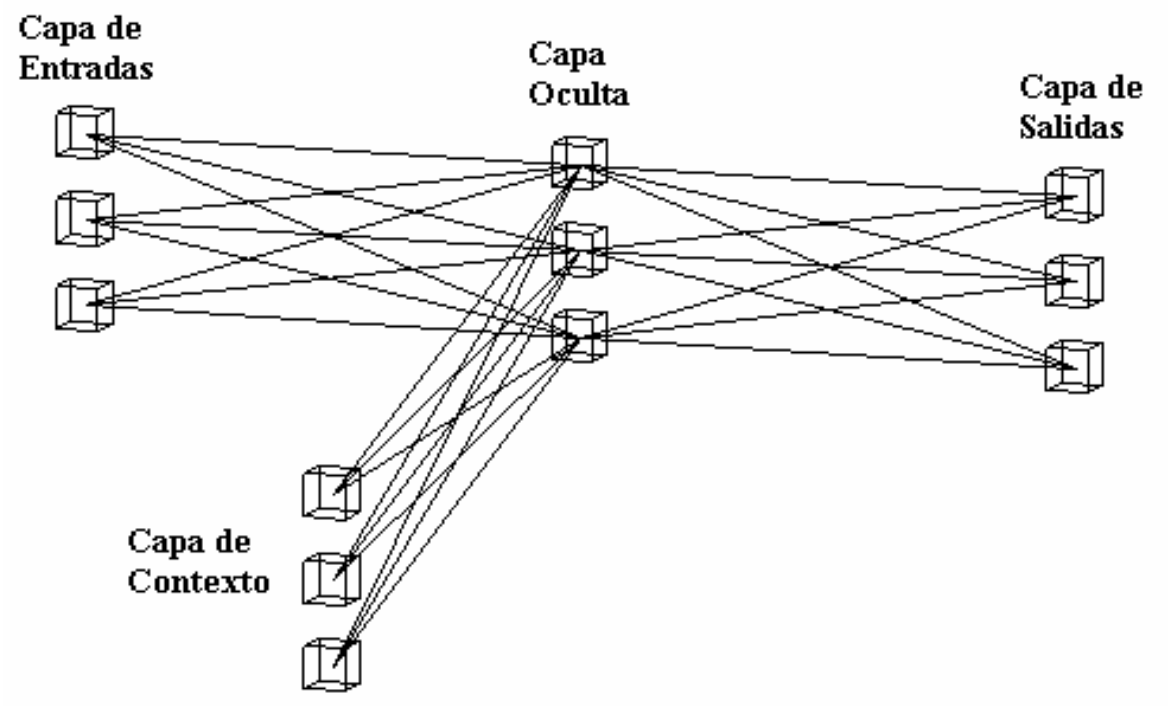

Figura 6. Red neuronal Elman

6.5 CARACTERISTICAS DE LAS REDES NEURONALES ARTIFICIALES

Las Redes Neuronales Artificiales (ANN) están compuestas por elementos que se comportan de forma parecida a las funciones más elementales de una neurona biológica. Estos elementos se organizan de una manera que puede o no estar relacionada a la manera en que está organizado el cerebro. Sin embargo las ANN presentan algunas características propias del cerebro, como lo son:

- Aprender de la experiencia (Aprendizaje).
- Generalizar basado en ejemplos previos (Generalización).
- Abstraer características esenciales de las entradas que contienen datos irrelevantes (Abstracción).

6.5.1 Aprendizaje

Las ANN pueden modificar su comportamiento en respuesta al medio ambiente. Las ANN se auto-ajustan produciendo respuestas consistentes con el medio ambiente. Existen muchos algoritmos de aprendizaje que pueden aplicarse a las ANN.

6.5.2 Generalización

Una vez entrenada, una ANN es, hasta cierto grado, insensible a variaciones pequeñas en sus entradas. Esto es, las ANN producen sistemas capaces de manejar el mundo "imperfecto" en que vivimos.

6.5.3 Abstracción

Las ANN son capaces de abstraer la esencia de una serie de entradas. Se pueden abstraer patrones perfectos de modelos distorsionados.

7. PROTOTIPO IDS

7.1 DESCRIPCIÓN

Los estudios actualmente desarrollados sobre el análisis de comportamientos para detección de intrusos, han sido desarrollados para analizar campos muy genéricos de datos. El enfoque inmunológico demuestra la necesidad de sistemas multiniveles donde cada nivel debe especializarse en algo. Así la detección debe enfocarse en un tipo de dato externo al sistema protegido para evitar que al fallar el proceso de detección, la infección ya este en el sistema.

Dicho dato externo debe analizarse al nivel correcto para poder analizar características propias del protocolo, las cuales se pierden al analizar en niveles más bajos (ej. llamados al sistema). Por lo tanto, el nivel propicio de análisis es el nivel mismo del protocolo. Además, este análisis esta determinado por patrones de comportamiento, para los cuales las redes neuronales artificiales son bastante útiles.

Adicionalmente el análisis de comportamientos debe estar enfocado hacia la detección de anomalías (anomaly detection) para poder detectar nuevos ataques por medio de algoritmos basados en redes neuronales como las redes Elman que demostraron su efectividad para la detección de anomalías [7].

El sistema propuesto no se enfoca en un punto determinado o en un paradigma determinado. Aprovecha las características que otros sistemas han desarrollado. Así teniendo en cuenta el enfoque inmunológico, el sistema multinivel tendría su piel como un filtro de características anómalas.

Tan importante como es la detección, lo es la identificación de los ataques. Por lo tanto, inicialmente el último nivel del sistema estaba diseñado de igual manera que el de detección, pero entrenado para la clasificación básica de los ataques conocidos, aunque al final se agrupo estos 2 niveles en uno solo, realizando una detección por identificación. La combinación de detección e identificación disminuiría sustancialmente, como así sucedió, los falsos-negativos y falsos-positivos presentes, con la capacidad de detectar ataques no conocidos y de identificar ataques comunes. Así se aprovecharía las ventajas y las limitaciones de los múltiples enfoques comúnmente utilizados y estudiados en la actualidad.

En la Figura 7. Se presenta la arquitectura lógica y física del sistema inicial propuesto.

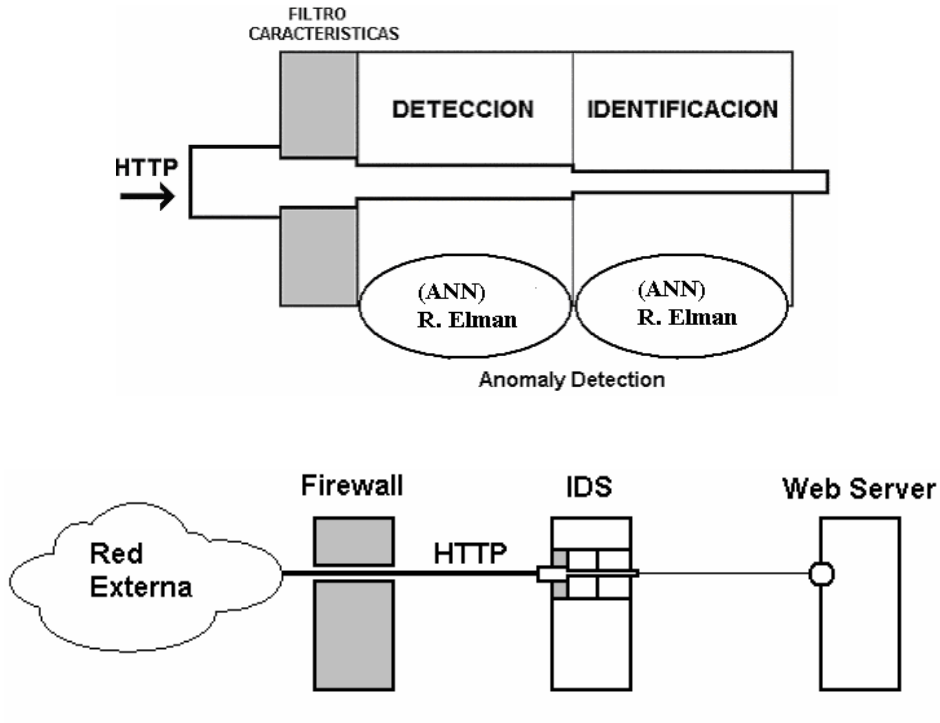

Figura 7. Sistema inicial propuesto

7.2 HERRAMIENTAS

El siguiente es un listado de las herramientas utilizadas para el desarrollo del prototipo.

7.2.1 Herramientas Hardware

- Procesador: Pentium IV Intel 2 GHz.
- Memoria: 256 Gbytes
- Disco Duro: 10 Gbytes

7.2.2 Herramientas Software

La totalidad del software a utilizar corresponde a herramientas de libre distribución en Internet o que hacen parte normal de la distribución del sistema operativo Linux.

Principales componentes de software:

- SNNS (Stutgart Neural Networks Simulator)
  Simulador para el diseño, creación, entrenamiento y generación de código fuente de redes neuronales.
- JNNS (Java Neural Network Simulator)
  Simulador Java para la creación y entrenamiento de redes neuronales.
- Compilador GCC.
  Compilador para lenguaje C.
- Sistema Operativo Linux (Mandrake 9/ Redhat)
- SQUID Web Proxy Cache Server v2.5 STABLE 2.

- Xplot (www.xplot.org). jPlot (Version java)
  Graficador de datos.
  (http://irg.cs.ohiou.edu/software/tcptrace/jPlot/)

7.3 ARQUITECTURA

Los IDS a nivel de red (NIDS) tienen su gran limitación en el nivel de captura de datos y en la tecnología utilizada para la obtención de los mismos a este nivel. Entre sus limitaciones se encuentra el no poder capturar y analizar todo el tráfico que debe ser observado. Esta limitación proviene de los mismos analizadores de tráfico (sniffers), los cuales son los sensores del sistema. El sistema tiene que manejar problemas de red como la fragmentación de datos, lo cual implica utilizar poder de procesamiento en la manipulación de datos, bajando el rendimiento del sistema y la probabilidad que detecte un ataque. Además, debido a que se están tomando los datos directamente del nivel de red, el factor tiempo, se convierte en una técnica anti-ids ya que se pueden realizar ataques separados por franjas de tiempo superiores a las que puede manejar el NIDS [12]. El panorama es mas sombrío si los datos a ese nivel están codificados con una técnica anti-ids o simplemente encriptados (HTTPS) en este caso el NIDS es totalmente inservible. Adicionalmente el sensor debe estar estratégicamente posicionado y obtener la totalidad del tráfico a analizar, cosa que en una red con switches puede ser complicado, debido a la segmentación que realizan estos dispositivos, evitando que se pueda capturar la totalidad del tráfico en la red.

Teniendo en cuenta las limitaciones mencionadas, el primer problema consiste en definir una arquitectura que elimine las limitaciones actuales y ofrezca ventajas adicionales. Así la arquitectura elaborada se basa en un Proxy inverso (reverse Proxy), lo cual elimina los problemas directamente relacionados con el nivel de red para captura de datos, permite obtener la totalidad de los datos dirigidos hacia los servidores Web, discriminando automáticamente los otros

protocolos y si estos datos están encriptados en este punto pueden ser desencriptados y analizados antes de ser redirigidos hacia los servidores Web protegidos en la zona desmilitarizada (DMZ) o zona de servicios públicos. Debido a que la manipulación de los datos se realiza a nivel de aplicación no existe una fragmentación de estos y por lo tanto pueden ser procesados, analizados correctamente y tomar decisiones que convierten al sistema en un sistema de prevención de intrusos.

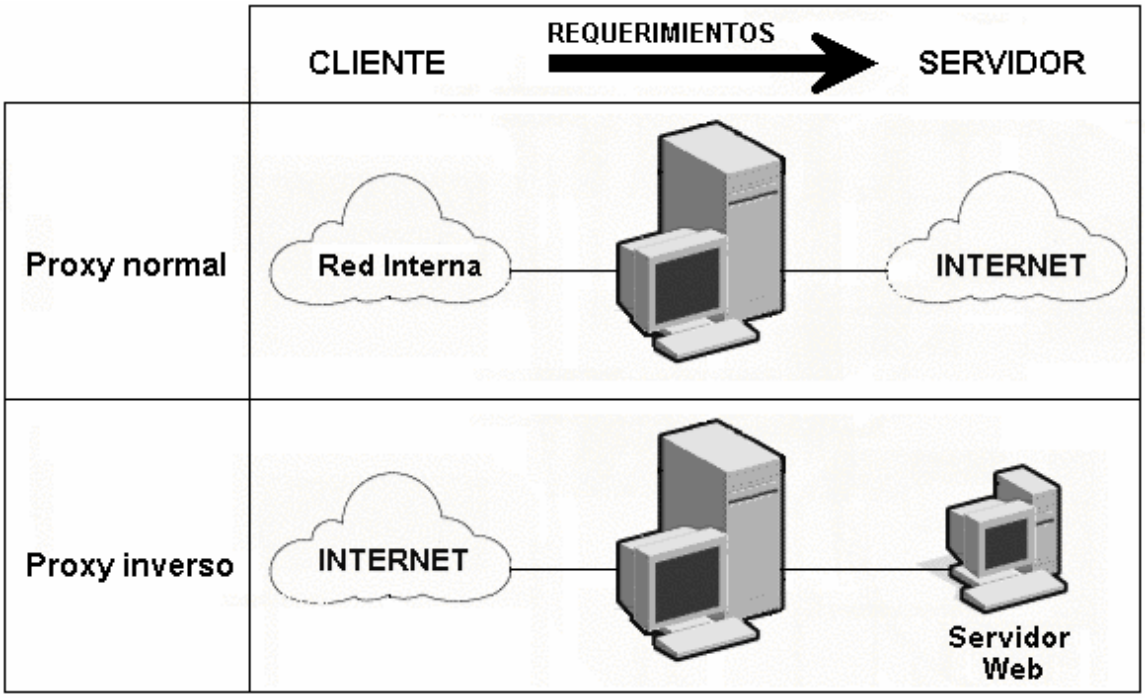

Figura 8. Proxy normal vs. Proxy inverso

Adicionalmente, al separar el nivel de análisis y detección en un Proxy inverso, se elimina el riesgo inherente de la complejidad adicional generada por un firewall de aplicación y eliminamos la dependencia o problemas de

compatibilidad del IDS con respecto a los servidores Web protegidos y las arquitecturas de sus sistemas. Básicamente el sistema IDS se convierte en un sistema genérico para la protección de cualquier tipo de servidor/aplicación Web.

El uso de un Proxy inverso como mecanismo de seguridad no es nuevo [15]. Algunos justifican su uso por convertirse en un único punto de entrada hacia servidores Web y principalmente por que esconde el direccionamiento IP interno. Lo cual es totalmente falso, ya que el esconder el direccionamiento interno, como lo hace un NAT, solo es funcional y efectivo a nivel de seguridad cuando cualquier requerimiento proveniente de una red insegura (Internet), no puede alcanzar a los hosts protegidos identificados con IPs inválidos. En un Proxy inverso los requerimientos provenientes de Internet siempre alcanzan los servidores Web protegidos aunque tengan IPs inválidas. El manejo de un Proxy inverso como único punto de entrada es eficiente solo si se hace algún tipo de análisis o tratamiento a nivel de seguridad sobre los requerimientos, lo cual hasta el momento solo sigue el mismo enfoque estático e ineficiente de firmas, para la posible detección de intrusos.

Para el desarrollo del prototipo actual se eligió como base de implementación el Proxy SQUID [16] que hace parte de la comunidad de software de código abierto. SQUID es un servidor Proxy que puede ser configurado como Proxy inverso (Reverse Proxy). Está diseñado para correr sobre sistemas Unix bajo la licencia pública general GNU (GPL).

7.3.1 SQUID Como base para el desarrollo de sistemas IDS para protocolo HTTP

La selección de SQUID como base para la implementación del prototipo está basado en sus características funcionales, las cuales han sido aprovechadas

para el desarrollo de filtros y análisis de contenido en protocolo HTTP por medio de su interfaze redirectora (redirector).

7.3.1.1 Redirector

El redirector es un programa que es ejecutado por SQUID cuando un requerimiento llega al Proxy [17]. SQUID redirecciona el requerimiento hacia el redirector y espera una respuesta. El redirector debe ser un programa continuo que nunca termina. Este es generalmente utilizado para analizar y filtrar requerimientos, si el requerimiento hace referencia a un destino no autorizado, o contiene una cadena no válida, el redirector retorna un requerimiento HTTP, el cual se convierte en el nuevo destino del requerimiento inicial, este puede ser ahora una página con un mensaje de error o simplemente el requerimiento inicial si no se determinaron problemas.

SQUID entrega al redirector, por medio de la entrada estándar, una cadena con el siguiente formato:

URL ip-address/fqdn ident method

Proyectos como Jeanne [18], utilizan este mecanismo y a SQUID en modo de Proxy inverso, para analizar todos los requerimientos dirigidos hacia un servidor Web protegido. Si el requerimiento incluye cadenas de ataques, el requerimiento es negado. El primer acercamiento para el desarrollo del prototipo fue el de utilizar también, la interfaze redirectora como punto de entrada al sistema IDS para luego procesar el requerimiento y generar una respuesta acorde. Desafortunadamente el nivel de detalle que entrega la interfaze redirectora de SQUID al redirector es muy pobre. Por ejemplo si un requerimiento es realizado a un CGI por medio del método GET, el redirector recibirá la totalidad del URL incluyendo los parámetros y sus valores enviados hacia él. Sin embargo si el

CGI recibe sus datos por medio del método POST el redirector solo recibirá el URL sin el cuerpo (BODY) el cual contiene todos los parámetros y sus valores. Si se tiene en cuenta que un ataque puede estar incluido en cualquier parte del requerimiento, este mecanismo es totalmente inadecuado para realizar este tipo de tareas, a no ser que se realizase una modificación al código fuente de SQUID para proveer la totalidad del requerimiento al redirector.

SQUID recibe sus requerimientos a través de las rutinas especificadas en client_side.c, cada requerimiento es manipulado por medio de las estructuras definidas en structs.h y es por medio de la estructura ConnStateData por la cual redirect.c crea la cadena que es entregada al redirector configurado. En principio el requerimiento podría ser obtenido en redirect.c directamente de la estructura ConnStateData, sin embargo SQUID reutiliza los buffers de dicha estructura, sobrescribiendo el requerimiento cada vez que es procesado en niveles superiores. Adicionalmente a este nivel el requerimiento ha sido decodificado eliminando información vital para su análisis.

Teniendo en cuenta esta limitación, la necesidad de obtener los requerimientos no procesados y la necesidad de realizar el análisis de los datos por medio de la interfaze redirectora, para manipular la respuesta y minimizar el impacto que genera la modificación de la arquitectura de SQUID, se realizó un puente entre client_side.c y redirect.c por medio de la estructura _ConnStateData. Así cuando un requerimiento es recibido por la rutina clientReadRequest() definida en cllient_side.c este es copiado en su totalidad al buffer total_request el cual se creó en la estructura _ConnStateData y puede ser obtenido por redirect.c para analizar el contenido en su totalidad. También se decidió realizar todas las actividades de análisis y procesamiento del IDS en la rutina redirectStart() de redirect.c y las de respuesta en el redirector asociado. Esto debido a que SQUID genera múltiples procesos del redirector asociado, lo cual puede disminuir el desempeño del servidor, teniendo en cuenta el nivel de análisis y procesamiento desarrollado en el IDS.

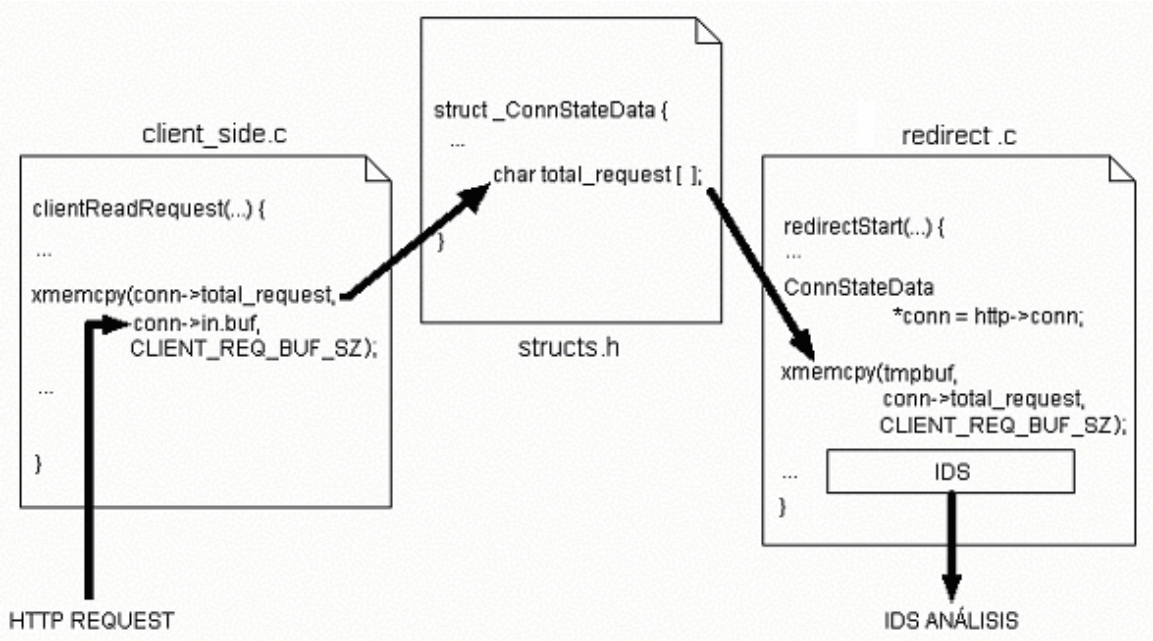

Figura 9. Modificaciones a SQUID

NOTA: El prototipo IDS fue elaborado directamente en la función redirectStart() modificando el código fuente de redirect.c él cual hace parte de la arquitectura básica de SQUID. El código realizado sigue los lineamientos a nivel de diseño e implementación planteados y desarrollados en este documento.

7.3.1.2 Manejo de requerimientos

Con las modificaciones realizadas, el manejo de requerimientos sigue igual aunque el detalle de estos es superior al nivel del redirector. Cuando un nuevo requerimiento llega éste es copiado al buffer total_request, y luego manipulado, decodificado y procesado por SQUID. Posteriormente si existe un redirector asociado, SQUID pasa a la rutina redirectStart() de redirect.c donde es tomado por el IDS la totalidad del requerimiento, analizado y procesado, generando una respuesta la cual es a su vez entregada al redirector asociado para decidir cual es el camino a tomar según el análisis del IDS.

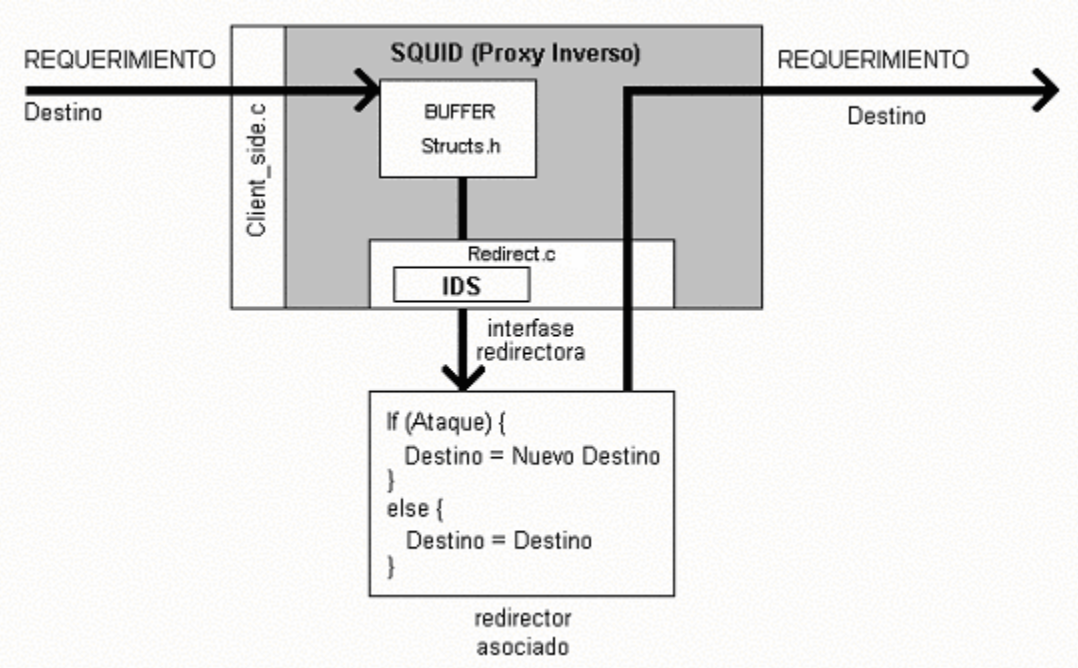

Figura 10. Manejo de requerimientos

Según lo anterior, el IDS recibe el requerimiento después de cierto nivel de procesamiento por parte de SQUID. Esto convierte automáticamente a SQUID en un filtro estructural de requerimientos, los cuales deben cumplir con la estructura definida en HTTP/1.1. Lo anterior, genera una limitación clara. Si existe un problema de seguridad grave en la implementación de las rutinas iniciales de SQUID este podría ser vulnerado y si el IDS es implementado en la entrada del Proxy se pierde la posibilidad y facilidad de manipular la respuesta, a no ser que se realicen mayores modificaciones con un alto impacto en la arquitectura inicial de SQUID.

Las limitaciones que impone la arquitectura de SQUID para analizar la totalidad de los datos incluidos en un requerimiento, muestran que en la actualidad

SQUID, sin ser modificado, no es adecuado para desarrollar actividades de análisis y filtrado de contenido para la protección de servidores Web, pues el nivel de detalle de los datos manejados por la interfaze redirectora es muy bajo.

7.3.2 Arquitectura de Red

Adicional a la arquitectura interna del IDS incluido en el Proxy inverso, es importante determinar la arquitectura de red óptima para implementar el IDS, con respecto a una topología de red adecuada para incrementar los niveles de seguridad requeridos por una organización.

El Proxy inverso-IDS debe ser instalado en la DMZ de la organización. El servidor Web puede ser reposicionado en una subred aparte o en la misma DMZ, pero nunca en la red interna ya que el Servidor Web sigue siendo un servicio público al que finalmente se tiene acceso desde una red externa como Internet.

Sobre la configuración del Firewall, en el caso que el Proxy inverso-IDS este ubicado en otra subred distinta a la del servidor Web a proteger, el Firewall debe permitir el trafico desde Internet hacia el Proxy inverso-IDS, (Puerto TCP 80) y permitir el tráfico entre este ultimo y el servidor Web final (Puerto TCP 80). Si el servidor Web se encuentra en la misma subred que el Proxy inverso-IDS, solo es necesario permitir el tráfico desde Internet hacia el Proxy inverso-IDS (Puerto TCP 80).

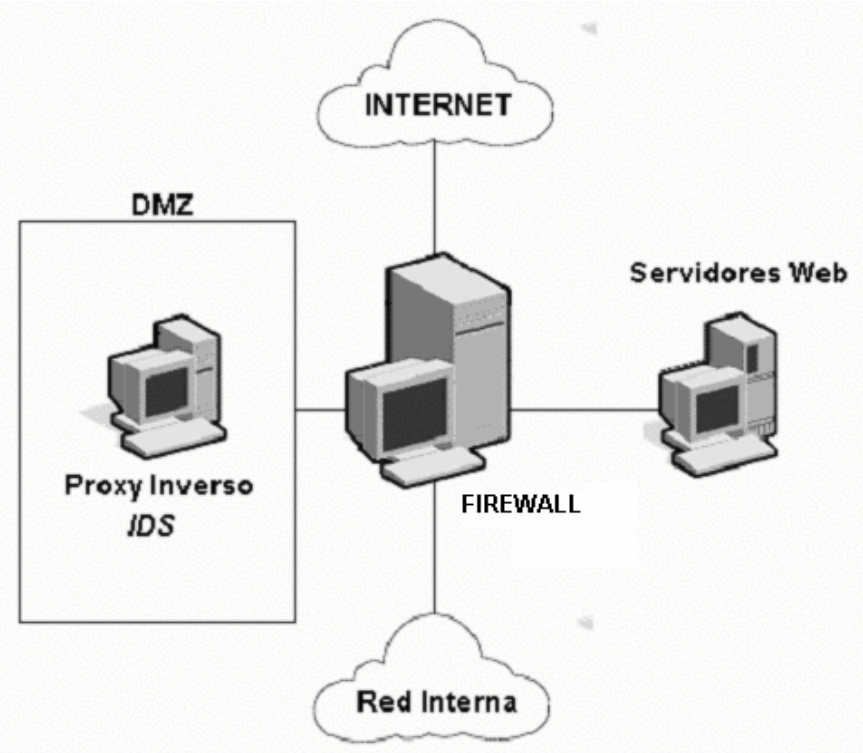

Figura 11. Arquitectura de red

7.4 SISTEMA DE DETECCIÓN MULTINIVEL

En seguridad informática, así como no existe un 100% de protección, no existen soluciones globales a los problemas de seguridad presentes. Debido a esto se han generado múltiples niveles de protección, donde cada nivel de protección es el encargado de hacer tareas propias y específicas. Eso significa que cada nivel se especializa en realizar su trabajo. Así el conjunto de protección y especialización por niveles, provee "niveles" de seguridad superiores, minimizando el impacto de un riesgo de seguridad existente. La clave de este enfoque esta básicamente en proveer un número adecuado de niveles o capas, balanceando e identificando plenamente las tareas que cada una debe cumplir (especialización) [19].

Sin embargo las investigaciones académicas sobre este tema tratan de generar soluciones globales para una amplia gama de problemas de seguridad. Es así como, por ejemplo un ataque del tipo de desbordamiento de pila o Stack Overflows (SoF), es tratado de la misma manera y al mismo nivel para su detección, como una inyección de código SQL (SQL injection). Aunque las 2 corresponden a vulnerabilidades de seguridad conocidas, ambas funcionan de forma muy diferente y por lo tanto necesitan de un tratamiento distinto. En un requerimiento HTTP que incluya una inyección de código SQL es factible reconocer ese tipo de ataque debido a que posee un patrón bien definido, sin embargo un SoF por sus características, permite ser codificado de tal forma que sea muy difícil identificarlo [20]. En consecuencia, el proceso de identificación de un SoF se convierte en una tarea igual o mayor a la de la detección de un virus polimórfico [21].

Así la teoría de utilizar múltiples niveles de protección y detección es replanteada al diseño del proyecto donde el sistema desarrollado en el nivel de aplicación esta compuesto por 2 niveles. El primer nivel corresponde a un filtro, el cual es según el enfoque inmunológico la “piel” del sistema. Este nivel es el encargado de repeler aquellos requerimientos cuya longitud sobrepasa un límite normal predeterminado. El segundo nivel recibe los requerimientos del filtro, para ser analizados e identificar en ellos algún tipo de ataque.

7.4.1 Nivel de filtrado

El IDS no debe enfocarse en un paradigma determinado. Debe aprovechar las características que otros sistemas han desarrollado. Así teniendo en cuenta el enfoque inmunológico, el sistema multinivel tendría su piel como un filtro de características anómalas.

Este nivel es una adecuación lógica de las tareas realizadas por un firewall de aplicación. Este tipo de IDS monitorea la longitud y composición básica de los requerimientos específicamente determinados por un URL, lo cual en teoría reduce significantemente el riesgo del compromiso de un servidor.

Entre las características más importantes de filtrado se encuentra el limitar el tamaño de los diferentes elementos de datos permitidos de HTTP. Por ejemplo, entre los elementos de datos se encuentran:

Atributo	**Tamaño máx. recomendado (chars)**	
	www.eeye.com SecureIIS 1.2.1	**www.flicks.com Titan**
URL	1024	2047
GET Query	1024	2047
POST Query	10000	64000
Variable en Query	128	
Datos en Query	512	
Header	1024	511
Accept	256	
Referer	256	
Accept-Lenguage	256	
Accept-Encoding	256	

User-Agent	256	
Host	256	
Connection	256	
Cookie	256	
If-Modified-Since	256	
If-None-Match	256	
Authorization	256	

Figura 12. Atributos vs. Tamaño recomendado

No existe evidencia pública alguna sobre el mecanismo o bases en la determinación de los valores recomendados por parte de los fabricantes de estos y otros productos [22]. Sin embargo por los tamaños de estos límites, se da a entender que este mecanismo de protección contra stack overflows esta dirigido a proteger directamente el servidor web más no las aplicaciones desarrolladas sobre él (ej. CGIs).

Que sucedería si el servidor web es susceptible a un buffer overflow en el encabezado “Host” con aproximadamente 257 caracteres y adicionalmente un CGI desarrollado utiliza este mismo campo, pero en su desarrollo quedo vulnerable a un buffer overflow con 60 caracteres? Al implementar el firewall de aplicación el servidor Web no sería vulnerable al primer BoF (Buffer Overflow) pero si sería vulnerable sobre el segundo.

Es muy difícil determinar con exactitud los límites máximos recomendables de los atributos, la dificultad radica en que cada servidor Web es único. Único en sus usuarios y único en el tipo de aplicaciones que ofrece. Por lo tanto es necesaria una herramienta que permita determinar los valores apropiados a

cada ambiente. Esta herramienta estadística notifica en tiempo real datos como el tamaño máximo de cada uno de los atributos y su porcentaje de uso. Con estos datos es mucho más factible determinar, por ejemplo, que el tamaño del encabezado “Host” nunca sobrepasa los 45 caracteres y por lo tanto sería recomendable manejar un límite máximo de 45 caracteres en vez de 256 iniciales. Además, el porcentaje de uso de los atributos permite saber con exactitud la necesidad de permitir solo los atributos (ej. Métodos) que en realidad se están usando, dejando por fuera muchas otras opciones que pueden contener problemas de seguridad.

Un problema de seguridad puede ocurrir en cualquier atributo manejado, sin importar el tipo de vulnerabilidad (heap overflow, buffer overflow, format bug, etc.) Por lo tanto el limitar el uso de atributos no debe estar circunscrito a solo algunos. Todos aquellos parámetros con sus valores contenidos en el URL deben ser filtrados. Los cuales con ayuda del análisis estadístico de su uso y longitud, puede determinarse con exactitud si es necesario permitir su uso.

El mecanismo de limitar las longitudes de los parámetros y sus valores, es una herramienta que permite contrarrestar la gran mayoría de problemas relacionados con SoF, sin embargo los firewalls de aplicación actuales proveen un mínimo de protección sobre estos, al nivel de las aplicaciones soportadas sobre los servidores Web supuestamente protegidos. Adicionalmente al limitar las longitudes de los parámetros en un requerimiento, basado en información estadística capturada en tiempo real y ajustada por el administrador del sistema, este nivel también identifica la presencia de cadenas comunes que pueden significar algún intento de ataque, por ejemplo “/etc/passwd” o “/repair/sam._”.

7.4.1.1 Codificación de requerimientos

Entre las técnicas más comunes para la evasión de IDS de cualquier tipo, se encuentra la codificación de datos. Esta codificación esta basada en la incapacidad del IDS para obtener el significado real de los datos obtenidos y por lo tanto no pueden ser analizados. El caso más común de codificación es la encripción de los mismos. Es por esto que un IDS a nivel de red no puede analizar el tráfico HTTPS (Secure HTTP). Para el caso del IDS desarrollado, si se requiere analizar tráfico HTTPS, SQUID puede ser configurado para recibir el requerimiento y desencriptarlo, debido a que el Proxy inverso es para el cliente el servidor Web. Posterior a su análisis se redirecciona el requerimiento a su destino final.

La codificación hexadecimal y unicode de los caracteres de un requerimiento corresponde a las técnicas más comunes utilizadas para la evasión de IDS. Debido a esto el IDS decodifica los caracteres que se encuentren representados por su valor ASCII hexadecimal y luego si es procesado. Sin embargo, el analizar que cada parte de un requerimiento se encuentra dentro de los límites de tamaño especificados, debe hacerse previo a cualquier decodificación pues el tamaño de un requerimiento codificado por estos medios es mayor a su tamaño decodificado.

Para determinar la presencia de cadenas que indiquen un ataque, si es necesario realizar previamente la decodificación de los requerimientos, para evitar precisamente que dichas cadenas estén codificadas en el requerimiento evitando su detección. Técnicas adicionales a la codificación de datos, corresponden a la modificación de la estructura de los mismos. Estas técnicas son eliminadas por el nivel de detección.

7.4.2 Nivel de detección

Uno de los principales objetivos de este nivel, es poder detectar ataques previamente no conocidos. Así, al analizar una muestra de 3000 problemas de seguridad reportados desde 1994 en servidores Web, CGIs y aplicaciones afines [23], se pudo evidenciar, con preocupación, que los problemas de seguridad en protocolo HTTP se repiten y básicamente corresponden a:

1- Problemas de seguridad previamente publicados pero que se presentan de igual forma en otra aplicación o servidor Web.
2- Variaciones de un problema de seguridad previamente publicado.
3- Conjunto de varios problemas de seguridad previamente publicados.

En conclusión, no existen técnicas radicalmente nuevas en el arsenal para la explotación de problemas de seguridad en servidores Web, CGIs, o toda aquella aplicación que trabaje con el protocolo HTTP. Sin embargo siguen apareciendo.

Teniendo en cuenta esto, la detección de ataques no conocidos está basado en independizar el nombre, tipo de aplicación o parámetro vulnerable, de la vulnerabilidad en sí. Adicionalmente, un mecanismo que pueda generalizar basado en patrones de ataques básicos, previamente aprendidos, puede generalizar para detectar nuevos. Este tipo de mecanismo lo ofrecen las redes neuronales artificiales. Una red neuronal artificial Elman [11] es el núcleo principal del nivel de detección.

La decisión sobre el tipo de red neuronal a utilizar se basó en estudios previos, los cuales demostraron su efectividad para la detección de patrones de ataque en un conjunto de datos (llamados al sistema) [1] [8] [9] y su uso en el reconocimiento de patrones en el lenguaje [11]. Es necesario recordar que un protocolo de comunicación, tal como lo es HTTP/1.1 es en si un lenguaje adoptado. Adicionalmente la red neuronal Elman maneja un nivel de memoria (contexto) la cual tiene en cuenta al procesar una entrada, las entradas previas.

Las entradas provienen directamente del nivel de filtrado y son procesadas por la red neuronal Elman para identificar un ataque en el requerimiento.

7.5 DETECCIÓN, IDENTIFICACIÓN Y CLASIFICACIÓN DE ATAQUES

Tan importante como lo es la detección de un ataque, lo es también o más, su identificación dentro de una clasificación dada. El poder identificar un ataque dentro de una clasificación provee información adicional a las personas encargadas de la seguridad en una organización y tomar las decisiones adecuadas según el tipo de ataque presentado. La detección es algo implícito en la clasificación, es por eso que la red neuronal desarrollada clasifica los ataques, como mecanismo de detección e identificación.

El principal problema que se presentó fue el de determinar una nueva clasificación, que fuese efectiva, sencilla, clara y con límites bien definidos. Al revisar múltiples clasificaciones de ataques presentadas en investigaciones semejantes, se pudo establecer que las clasificaciones utilizadas en la gran mayoría de los casos no tienen sus límites bien definidos [24] [25], así un ataque puede estar incluido en 2 o mas categorías al mismo tiempo. Y teniendo en cuenta que esta clasificación corresponde a los datos de salida de una red neuronal, puede generar no solo problemas en la identificación de ataques, también en su entrenamiento. La gran mayoría de investigaciones basan su clasificación en los efectos directos de la vulnerabilidad, lo cual es totalmente erróneo. Por ejemplo, una clasificación que maneje, Negación del Servicio (DoS), Ejecución de comandos y BoF, complica la identificación de este último ya que en el caso que el ataque falle, puede generar un DoS (ej. sobrescribir la dirección de retorno con un valor arbitrario), y si es efectivo puede ejecutar comandos del sistema o ejecutar su propio programa mas complejo especificado en su shellcode. Quizás este tipo de clasificaciones provienen de la clasificación

común que se hace de ciertas vulnerabilidades en el momento de su publicación en listas de correo especializadas.

Además, Teniendo en cuenta las observaciones previamente hechas (ver. 7.4.2) sobre los problemas de seguridad en protocolo HTTP, analizando nuevamente los ataques publicados y logs con tráfico normal, es factible la clasificación clara, sencilla y efectiva de los problemas de seguridad, basados en su causa y no en su efecto, generando así, la siguiente clasificación:

1. Cross Site Scripting (XSS).
2. Inyección de comandos.
3. Inyección de código SQL (SQL Injection).
4. Stack Overflows (SoF).
5. Manipulación de PATH.
6. NORMAL

Clasificación sencilla y completa, teniendo en cuenta que cualquier ataque solo tiene una posible clasificación. Lo cual facilita el entrenamiento de la red neuronal. Esta clasificación es resultado de este proyecto y por lo tanto corresponde a una nueva forma de ver las vulnerabilidades, particularmente en el protocolo HTTP.

La clasificación anterior posee la siguiente jerarquía, la cual puede ser ampliada según aparezcan nuevas tecnologías, lenguajes y aplicaciones dentro del mundo HTTP.

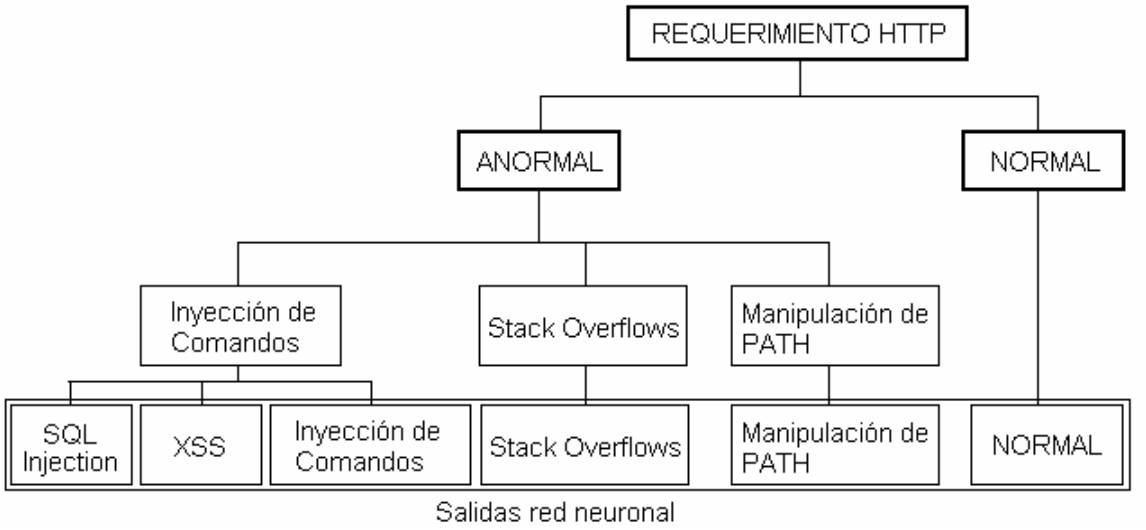

Figura 13. Clasificación de ataques

Existe un caso especial, el cual corresponde a aplicaciones con vulnerabilidades tan simples, como por ejemplo:

Prueba.cgi?file=/etc/passwd

En realidad esta aplicación, no posee en si una vulnerabilidad, ya que no se están modificando las características de las entradas (file) para modificar el comportamiento del CGI. Este CGI de ejemplo prácticamente está implementado para ofrecer el archivo. Seria contraproducente entrenar una red neuronal con este tipo de patrones ya que se estaría asociando una vulnerabilidad con un nombre de archivo. Este tipo de problemas son manejados en el nivel de filtrado, donde la ocurrencia de cadenas tales como "/etc/passwd" intuyen la presencia de un ataque pero no un tipo de vulnerabilidad asociado.

7.5.1 Inyección de comandos

Esta categoría de ataques es una agrupación que corresponde a la ejecución directa de comandos del sistema operativo o del lenguaje asociado, debido a

falencias en la validación de entradas a la aplicación, incluyendo el mismo servidor Web. Entre esta categoría se encuentra la ejecución de comandos del sistema operativo a través del shell, generalmente por medio de fallas en la implementación de llamados a rutinas que permiten la ejecución de comandos en el sistema como por ejemplo lo son en el lenguaje C, system(), exec(). Variaciones de este ataque dependen del tipo de ambiente donde se inyectan los comandos los cuales pueden ser en interfaces a consultas SQL y son lo que se denomina “SQL injection” o inyección de código SQL, para modificar el comportamiento de los queries generados. Si se inyecta código “Client-Side” como lo es HTML, java, javascript se genera un ataque del tipo “Cross site scripting” o XSS.

Como ejemplo clásico de este tipo de ataques se encuentra la ejecución remota de comandos a través del CGI PHF (1996) [32].

7.5.2 Manipulación de PATH

El PATH define la ubicación relativa o absoluta de uno o varios recursos para un contexto dado, por lo tanto la modificación de esta información permite manipular el comportamiento normal de las aplicaciones, al proveer ubicaciones distintas y por ende recursos distintos a los normalmente utilizados. El caso típico de este tipo de ataques corresponde al comúnmente llamado “directory transversal” o directorio transverso en el cual, por ejemplo, una aplicación entrega el contenido de un archivo dentro de un directorio raíz determinado. Si el nombre del archivo hace parte de las entradas manipulables por el usuario y si existe una falla en la validación de esta entrada, es posible acceder a archivos diferentes por fuera del directorio raíz, con entradas tales como:

Archivo = ../../../../../directorio_de_sistema/nuevo_archivo

Vulnerabilidades como la presentada por servidores Microsoft IIS con respecto a problemas en el tratamiento de caracteres Unicode, hace parte de esta clasificación [27].

Los problemas comunes en la implementación de aplicaciones en PHP, también se encuentran en esta categoría. La única diferencia en este caso es que el PATH utilizado no es interno al host atacado, por lo general, este es manipulado para hacer referencia a un host externo [28]. Adicionalmente, una gran cantidad de técnicas anti-IDS, que a su vez son ataques, basan su funcionamiento en la manipulación de PATHs y por lo tanto deben estar incluidos en esta categoría [29].

7.5.3 Stack Overflows

Ataques de este tipo han estado presentes y activos por décadas. Comúnmente se les denomina “Buffers Overflows” pero este es a su vez un tipo de vulnerabilidad dentro de esta categoría. Entre esta categoría se encuentran:

- Buffer Overflows [33]
- Off-by-one Overflows [34]
- Heap Overflows [35]
- Integer Overflows [36]
- Format Bugs [37]

Cabe anotar que aunque existe literatura que define esta categoría como propia a lenguajes como C o C++, y que existen lenguajes exentos de estos como Java [30], esto no es verdad [31] ya que este tipo de vulnerabilidades no depende directamente del tipo de lenguaje utilizado.

El concepto sobre el cual se basa este tipo de ataques, corresponde a la manipulación del contenido de la memoria, heap o stack (pila) para modificar sus datos, generalmente las direcciones de retorno. Para obtener el control sobre la ejecución de la aplicación y modificar su comportamiento, ejecutando instrucciones que permitan realizar acciones por fuera del curso normal del programa, como lo es la ejecución normal de comandos, el escalamiento de privilegios o causar la inestabilidad del sistema (DoS). Dicha manipulación se realiza causando un desbordamiento controlado del área de memoria del proceso afectado, sobrepasando los límites normales de los datos de entrada almacenados y controlados por el usuario. El sobrepaso de estos límites, es posible cuando no existe una correcta validación de los datos que se reciben.

La solución planteada e implementada en el prototipo esta basada en reforzar estos límites, teniendo en cuenta que los límites de los requerimientos normales (tráfico normal) están circunscritos dentro de los límites máximos que evitan un ataque como estos. Así, para los “Buffer Overflows” los límites están especificados por la longitud de las cadenas, mientras que para “Integer Overflows” están especificados por un rango. Esta solución esta encaminada hacia la protección, más que a la detección ya que esta ultima tarea es bastante compleja y mayor si se realiza externamente al host o servidor Web protegido. Este acercamiento es el mismo con el que algunos firewalls de aplicación han protegido con éxito algunos servidores Web [37] pero esta vez, utilizando valores límites que pueden ser ajustados según el tráfico normal de un servidor Web determinado, para no solo proteger el servidor directamente, si no también las aplicaciones sobre el, a través del Filtro de características (Nivel de filtrado) implementado en el prototipo.

7.5.3.1 Problemas en la detección de Stack Overflows

La detección se ha basado principalmente en encontrar patrones o firmas en las cadenas de ataque. Al principio simplemente se realizaba una búsqueda por cadenas, (representación hexadecimal) tales como "\x90\x90\x90…\x90" donde el código (Opcode) 0x90 representa la instrucción NOP (IA32) comúnmente utilizada para facilitar los métodos de explotación al tantear por la dirección de retorno al código que se desea ejecutar denominado "shellcode" [33]. Otras técnicas se enfocan en encontrar firmas en el "shellcode" como tal. "Shellcodes" son pequeños programas en lenguaje de máquina los cuales son especialmente diseñados y construidos para ser inyectados en la ejecución de un proceso vulnerable, para que este sea ejecutado con los privilegios otorgados inicialmente al proceso.

Una cadena típica de explotación generalmente está construida así:

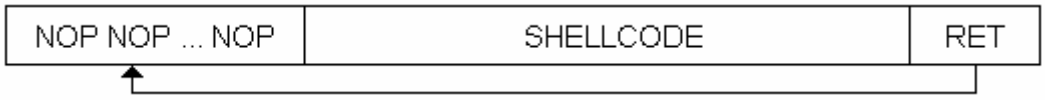

Figura 14. Buffer Overflow

Cuando se manipula un "Buffer Overflow" se genera este tipo de cadenas, las cuales son introducidas al buffer vulnerable para sobrescribir con la dirección de retorno (RET) la dirección de retorno original almacenada en la pila. Cuando se devuelve el control a la función, cuya dirección de retorno ahora fue modificada. Se ejecutará la secuencia NOP (No Operation) y luego el "shellcode" determinado, el cual comúnmente ejecuta el "shell". Si el proceso es privilegiado, o sea es ejecutado como root, el "shell" también quedará ejecutando comandos de forma privilegiada. Un "shellcode" típico para arquitecturas Intel (IA32) es:

```
char execshell[] =
"\xeb\x24\x5e\x8d\x1e\x89\x5e\x0b\x33\xd2\x89\x56\x07\x89\x56\x0f"
"\xb8\x1b\x56\x34\x12\x35\x10\x56\x34\x12\x8d\x4e\x0b\x8b\xd1\xcd"
"\x80\x33\xc0\x40\xcd\x80\xe8\xd7\xff\xff\xff/bin/sh";
```

(Tomado del exploit para mount v. 2.0.10 (linux). THC-magazine 3, 1996)

Adicional a detectar la presencia de cadenas con códigos NOP, también se busca la presencia de firmas que corresponden a opcodes comunes, como también la presencia de "/bin/sh" en ellos. Pero con la aparición de los sistemas IDS, los ataques también se han modificado para evitar su detección. Existen técnicas que evitan el uso de NOPs [39] o simplemente son reemplazados por alguna combinación de instrucciones que no realice ningún trabajo y en la mayoría de casos estos programas reutilizan las técnicas para el desarrollo de virus polimórficos (virus que autónomamente modifican su apariencia o forma en cada infección para evitar su detección) para que estos códigos cambien su estructura y puedan pasar inadvertidos, aunque para un shellcode es mucho mas fácil ya que se enfrenta con menos inconvenientes que un virus; por ejemplo un shellcode no tiene que preocuparse por mecanismos de infección y replicación [42], a no ser que esto último sea la finalidad misma del código, como sucede en algunos "gusanos". Estas pequeñas cadenas son programas y por lo tanto tienen múltiples combinaciones, formas y funcionalidades. Además, existen herramientas como JempiScodes [40] o ADM Mutate [41] que automatizan el proceso de generar shellcodes anti-ids, pudiendo producir billones de rutinas para la mutación [43]. Una posible solución para la detección seria el uso de técnicas de "sandboxing" (ejecutar un posible shellcode en un ambiente virtual protegido), pero implementar estas técnicas en host intermedios (Proxy inverso) además de complejo, podría traer problemas de rendimiento. En conclusión, la detección de este tipo de ataques, utilizando patrones, es una tarea poco eficiente y productiva si se tiene en cuenta que existe una gran cantidad de combinaciones posibles. Debido a esto la categoría de "Stack

Overflows" no es analizada en este nivel (nivel de detección), pero es manejada por el nivel de filtrado enfocándose en la protección.

7.6 DISEÑO Y ENTRENAMIENTO DE LA RED NEURONAL ELMAN

La generación de los datasets (datos para el entrenamiento de la red neuronal) se hizo utilizando las muestras de vulnerabilidades publicadas y bases de datos provenientes de algunos scanners de vulnerabilidades disponibles.

El primer acercamiento para el desarrollo de la red neuronal fue el de manejar entradas de dimensión variable para la etapa de entrenamiento. Este acercamiento presentó múltiples dificultades y poca efectividad, para lo cual se determinó el uso de entradas de dimensión estática en el entrenamiento y manejar una ventana corrediza para la presentación de los requerimientos (URL) en el momento de su funcionamiento normal.

Sobre la presentación de los datos, cada patrón, representado por una cadena de caracteres con tamaño fijo, se codificó cada carácter a un valor entre [0,1] traduciendo cada carácter a su valor entero ASCII y normalizando el vector, dividiendo cada valor sobre 255. Este tipo de codificación dificultó el aprendizaje, debido a que los valores generados por la codificación están tan cerca uno del otro que a la red neuronal le es difícil diferenciarlos [26] (Figura 14).

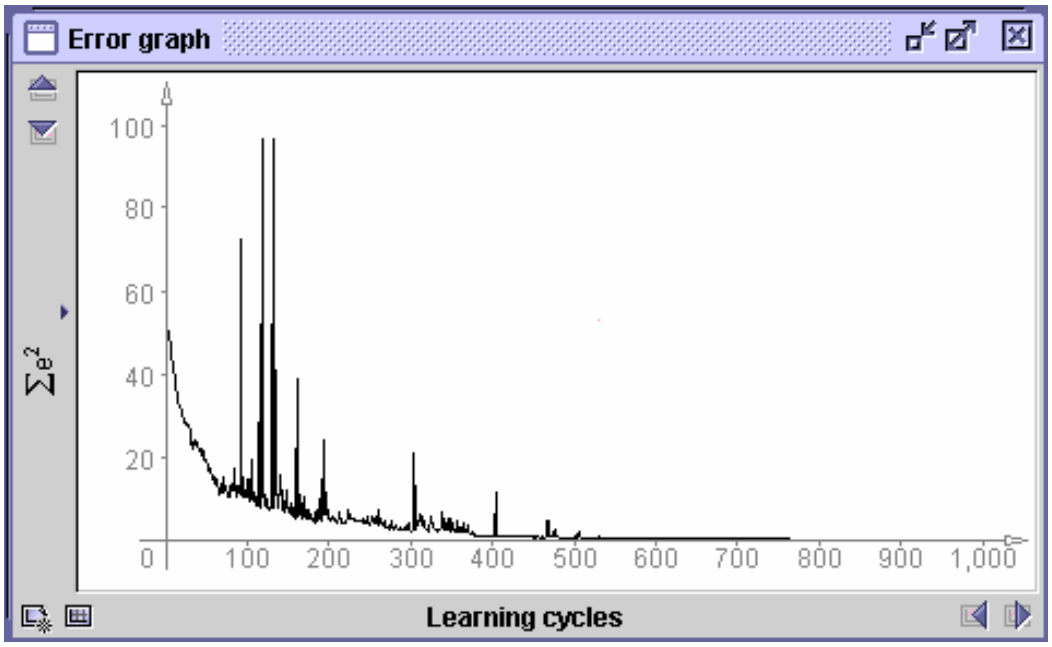

Figura 15. Entrenamiento codificación ASCII/255

Adicionalmente los *datasets* con esta codificación, fueron generados basados en patrones base obtenidos de la muestra inicial. Cada patrón base era procesado para generar nuevos patrones de entrenamiento incluyendo cadenas alfabéticas aleatorias. Esto a su vez generó *datasets* de entrenamiento muy grandes y poco efectivos.

Posteriormente, se desarrolló una codificación que permitiese la representación de un patrón en valores binarios. Esta codificación se basó en la representación binaria del valor ASCII de cada carácter, convirtiendo las entradas de la red neuronal en una matriz de longitud (8 x Longitud_ventana), lo cual genera una red neuronal de gran tamaño, difícil de procesar y con alto consumo de recursos computacionales. Aunque la red neuronal se comportó mucho mejor que los primeros modelos, como se observa en la Figura 15.

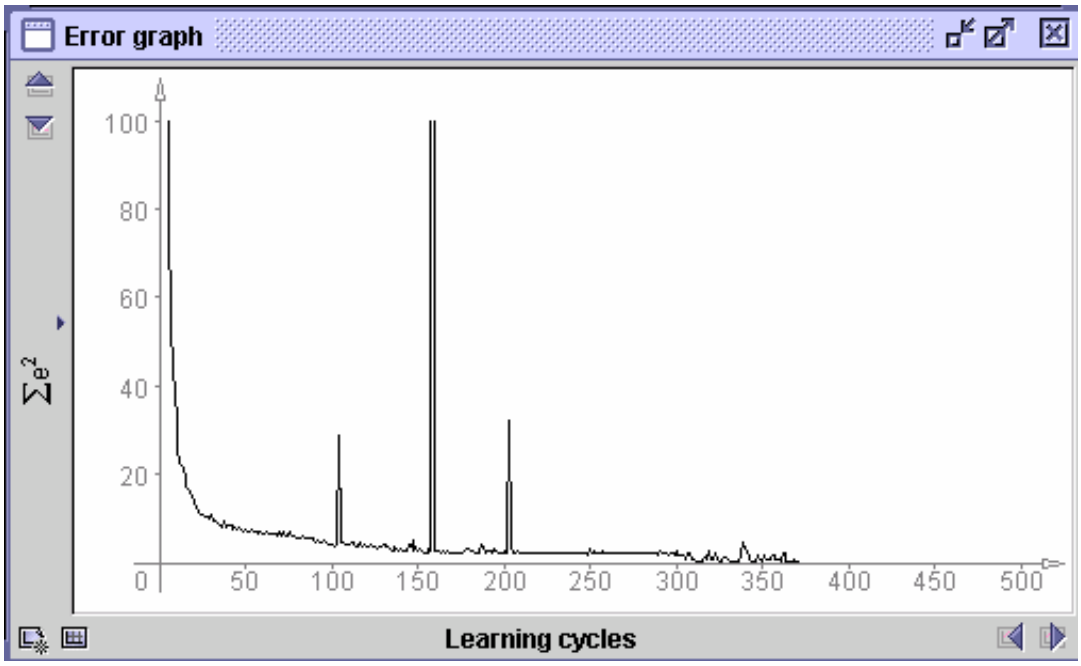

Figura 16. Entrenamiento codificación 8 bits

Para disminuir el tamaño de la red neuronal, se procesó a un nivel superior las entradas, a las cuales se le dio un valor de 255 a toda cadena alfanumérica. Dejando de lado las extensiones de programas, si existen y caracteres especiales, los cuales son los que al final determinan la presencia de un patrón de ataque. Así un URL es procesado y condensado en una estructura base. Por ejemplo:

Cadena:

nombre.exe?param1=..\..\archivo

Tipo de ataque:

Manipulación de PATH

Codificación:

(255)(46)(101)(120)(101)(63)(255)(61)(46)(46)(92)(46)(46)(92)(255)

Matriz 8 bits:

11111111
00101110
01100101
….
11111111

EL uso del valor 255 en vez de 0 como valor genérico, se basa en que el 0 es el valor ASCII del carácter NULL, el cual es comúnmente utilizado en la explotación de múltiples vulnerabilidades.

La decisión de mantener las extensiones de las aplicaciones, es debido a que algunas vulnerabilidades, aunque no dependen de la aplicación, dependen de su tipo, especificado en la mayoría de casos por su extensión. (Ej, manipulación del PATH de ubicación de librerías en PHP).

Esta codificación permitió eliminar la dependencia a los nombres de programas y enfocar el entrenamiento a patrones de ataques genéricos. Disminuyó el tamaño de la red neuronal así como el tamaño de los datasets de entrenamiento, los cuales ahora se componen básicamente de patrones genéricos codificados logrando que el tiempo de entrenamiento sea menor (Figura 16).

Ejemplo de dataset de entrenamiento (sin codificar a 8 bits, '@' = 255):

<Clasificación>;<patrón>
COMMAND_INJECTION;/@;@ /@/@|
COMMAND_INJECTION;;@ /@/@|?@
COMMAND_INJECTION;@?@=@&@=;@
COMMAND_INJECTION;@ /@/@|?@=
COMMAND_INJECTION;?@=@&@=;@%20

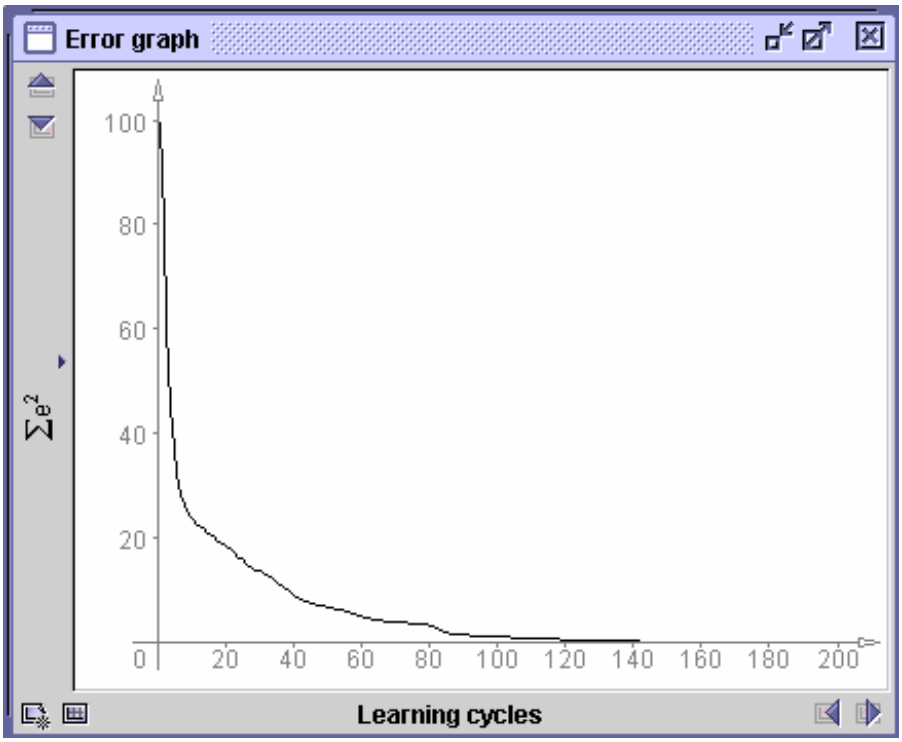

Figura 17. Entrenamiento codificación 8 bits normalizada y condensada

7.7 FUNCIONAMIENTO GENERAL DEL PROTOTIPO

Al llegar un requerimiento a SQUID este es redirigido a la interfaze redirectora en la cual se encuentra el IDS, allí la totalidad del requerimiento entra al nivel de filtrado donde es decodificado y se revisa por la presencia de las cadenas definidas en el archivo strings.conf (Archivo de configuración del prototipo que contiene las cadenas a revisar en los requerimientos). Si no está presente ninguna cadena, el requerimiento codificado es procesado comparando cada una de sus partes con los tamaños máximos y mínimos definidos en el archivo de configuración del IDS. Si cada parte del requerimiento cumple con los límites establecidos, el requerimiento es decodificado nuevamente y entregado al nivel de detección. En el cual se preprocesa el URL y el cuerpo (body) para ser analizados por la red neuronal. Si en la salida de la red existe un valor mayor a 0.9 este se toma como afirmativo en la concordancia con un ataque. La posición

del valor máximo identifica el tipo de ataque encontrado. Si la salida corresponde a tráfico normal o todos los valores son menores a 0.9 entonces el requerimiento pasa al destino final. Dependiendo de la configuración del IDS antes de ser reenviado a su destino final el requerimiento es procesado para determinar los tamaños de cada una de sus partes. Estos valores son almacenados en el archivo de auditoria limits.txt para su posterior análisis.

Si en alguna parte del IDS se encuentra un posible ataque este es almacenado en el log de auditoria, con la información relevante. Adicionalmente el IDS pasa al redirector asociado el código asociado del nivel donde se presentó una alerta o 0 si el requerimiento es totalmente normal. Esto para determinar el tipo de acción a tomar por parte del redirector para definir el destino apropiado según el caso.

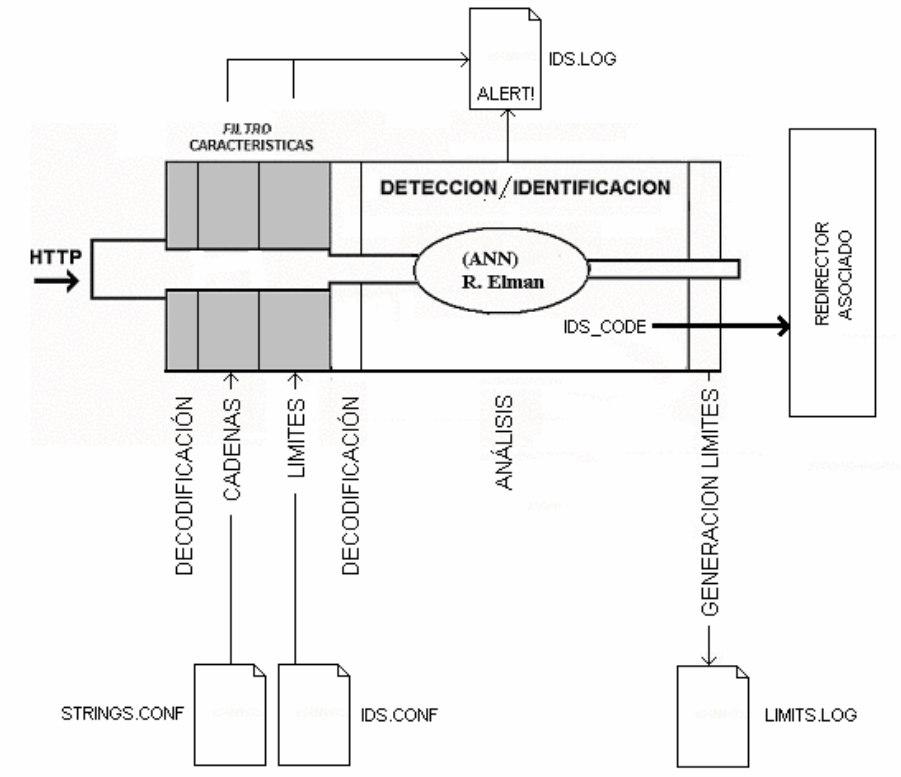

Figura 18. Funcionamiento general del prototipo

7.7.1 Redirector Asociado

El redirector asociado recibe de la interfaze redirectora de SQUID, por la entrada estándar el siguiente formato:

URL ip-address/fqdn ident method ids_code

El parámetro Ids_code es generado por el IDS y representa si el requerimiento proveniente de ip-address, después de ser analizado por el IDS, corresponde a un ataque o no. Si ids_code es diferente a 0, significa que es un ataque. Si es igual a 0 el requerimiento es normal y por lo tanto puede seguir su camino normal al servidor Web.

7.7.2 Posibles Acciones

Por lo general un NIDS normal, si identifica un ataque, solo genera una alarma. Esto se debe a la complejidad que implica reaccionar ante un ataque al nivel de red con una arquitectura basada en sensores.

Una ventaja adicional a manejar los procesos de detección al mismo nivel de los datos procesados es el de poder tomar decisiones activas con respecto a los resultados del análisis. Gracias a la facilidad que provee el redirector de definir el destino final de un requerimiento y teniendo en cuenta el valor de ids_code se pueden generar múltiples tipos de respuestas al descubrir un ataque:

- Si el destino final del requerimiento es el mismo servidor Web inicial el IDS solo se comportaría como un IDS normal al generar una alarma a nivel de log de auditoria.

- Si el destino final es el redireccionamiento a una pagina de error o la creación de una regla de filtrado prohibiendo toda conexión proveniente del ip de origen (ej. Netfilter/ iptables) el IDS esta tomando una acción mas activa y reactiva al ataque detectado.

- Si el destino final es un servidor Web alterno que personifique al original, este podría convertirse en un Honeypot transparente para el atacante.

El destino y la decisión sobre la respuesta a un ataque es decisión del redirector asociado. Esta flexibilidad de respuesta está facilitada por el nivel OSI en el que se manipulan y analizan los datos, lo cual hace que la implementación de un IPS (Intrusion Prevention System), como evolución de los IDS, para enfocarse mas en la prevención y no tanto en la detección de ataques, sea muy sencilla [44].

7.7.3 Análisis de limits.log

Al almacenar los tamaños de las diferentes partes que conforman un requerimiento, estos datos pueden ser analizados para ajustar la configuración genérica del IDS. Este ajuste permite proteger las aplicaciones que soporta el servidor Web y por ende el servidor mismo, ya que los límites que soportan las aplicaciones son iguales o menores a las que soporta su base. Si existe un buffer overflow en una de esta aplicaciones los límites obtenidos en un ambiente normal controlado son siempre menores a los necesarios para explotar esta vulnerabilidad, por lo tanto al estar bien definidos estos límites, el ataque será infructuoso. El archivo limits.txt contiene el tamaño de cada una de las partes

que conforman un requerimiento, el cual en casos normales, corresponde a los casos de uso de las aplicaciones implementadas. Estos tamaños son analizados automáticamente por la aplicación “idsconfig“, la cual genera el archivo de configuración adecuado para el IDS, este archivo incluye los mínimos y máximos de cada una de las partes de un requerimiento normal. Cabe recalcar que esta generación de configuración debe realizarse en un ambiente controlado para evitar la corrupción de datos y por lo tanto la corrupción de la configuración. Sin embargo “idsconfig” puede generar archivos que pueden ser graficados por la herramienta “Xplot” para poder ver mejor el comportamiento de los datos y descartar cualquier dato por fuera de los parámetros normales de uso. La siguiente gráfica muestra la representación de un ataque de buffer overflow sobre un servidor Web de pruebas.

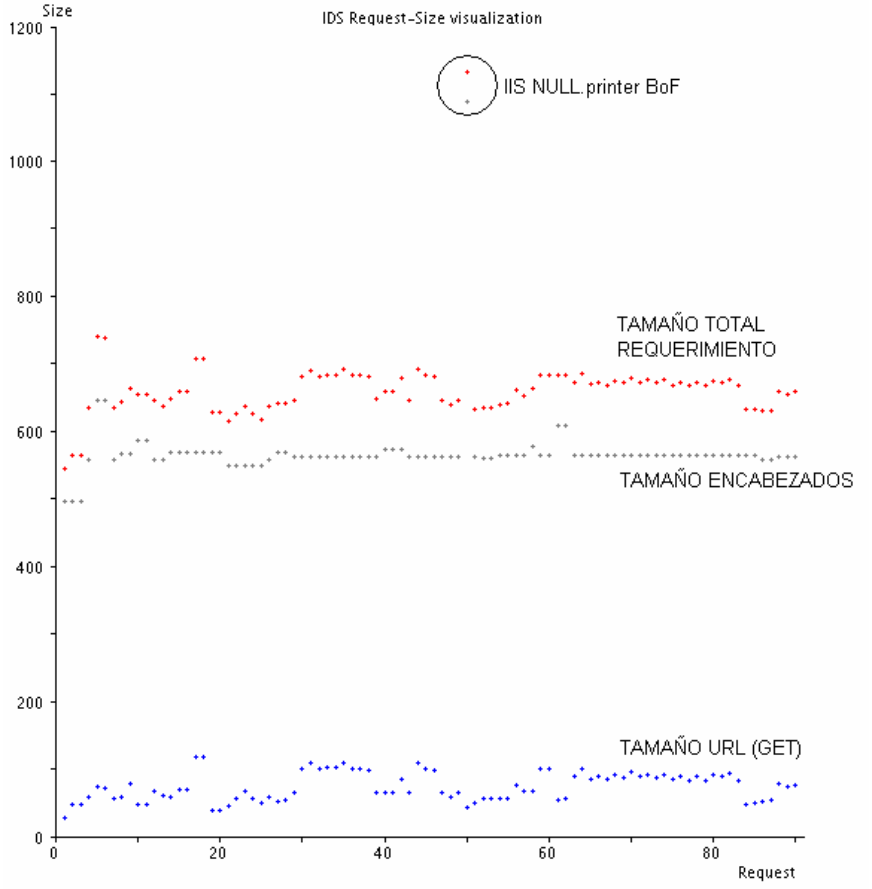

Figura 19. Análisis gràfico limits.log

8. PRUEBAS Y RESULTADOS

8.1 PRUEBAS

El desarrollo de las pruebas está basado en generar tráfico normal y ataques a un servidor Web simulado para obtener las siguientes mediciones:

- Porcentaje de efectividad total del sistema.
- Falsos positivos
- Falsos negativos.

Estas mediciones corresponden a las métricas para determinar la viabilidad del prototipo por medio de su correcto funcionamiento y su comparación con SNORT, aunque este tipo de comparaciones no hacen parte de los objetivos planteados para este trabajo, hacen parte de la continuidad presentada, de igual manera, mediciones de rendimiento están totalmente por fuera del alcance de este proyecto. La medición mas importante corresponde al porcentaje de detección total del sistema el cual incluye los falsos positivos y falsos negativos. En conclusión esta métrica corresponde al porcentaje de detección efectivo que determina si el sistema pudo detectar correctamente los ataques enviados. Adicionalmente existe un porcentaje de identificación el cual permite determinar la efectividad y por lo tanto la viabilidad de la red neuronal y su clasificación, este porcentaje corresponde al numero total de identificaciones correctas que realizó la red neuronal de los requerimientos enviados según la clasificación dada. Debido a que IDS como SNORT no poseen una clasificación comparable a la de la red neuronal, el porcentaje de identificación no es una métrica que pudiese ser usada para la comparación entre IDS, pero permite determinar su viabilidad. Las siguientes son la formulas utilizadas para generar los porcentajes:

Total Detectados = Normal + Ataques + Clasificación Errada

% de Detección = (Total Detectados * 100) / Total pruebas

Total Identificados = Normal + Ataques
% de Identificación = (Total Identificados * 100) / Total pruebas

Cada medición deberá ser analizada para determinar nuevas limitaciones del sistema y generar recomendaciones adicionales para su ajuste. Entre los resultados esperados se espera que las vulnerabilidades de diseño, no sean detectadas, debido a que este tipo de vulnerabilidades son explotadas con requerimientos que generalmente corresponden a tráfico normal. Sin embargo se espera un porcentaje de detección bastante alto para los requerimientos que traten de explotar problemas en la implementación de aplicaciones y servidores Web.

El típico ejemplo de una vulnerabilidad de diseño corresponde a una aplicación que esta diseñada con una estructura tal que sus archivos de configuración y datos están dentro del directorio raíz del servidor Web y por lo tanto pueden ser accesados directamente por un usuario realizando un requerimiento normal a dichos archivos.

Para la realización de las pruebas se utilizaron scanners de seguridad de código abierto (nikto, nessus) que cumplieron las siguientes condiciones.

- Posean pruebas a nivel de protocolo HTTP.
- De verdad realicen ataques, y que no solo verifiquen la existencia de un archivo o aplicativo posiblemente vulnerable.

Nikto: (http://www.cirt.net/code/nikto.shtml)

Nikto es un escáner de servidores web que busca más de 2000 archivos/CGIs potencialmente peligrosos y otros problemas en más de 200 servidores. Utiliza la biblioteca LibWhisker (Libwhisker es una biblioteca para perl que permite la creación de escáners de HTTP a medida) y su base de datos de vulnerabilidades es actualizada constantemente.

Nessus: (http://www.nessus.org/)

Es la herramienta de evaluación de seguridad "Open Source" de mayor renombre. Nessus es un escáner de seguridad remoto para Linux, BSD, Solaris y Otros Unix. Está basado en plug-in(s), tiene una interfaz basada en GTK, y realiza pruebas de seguridad remotas en diferentes protocolos. Permite generar reportes en HTML, XML, LaTeX, y texto ASCII; también sugiere soluciones para los problemas de seguridad.

La gran mayoría de scanners, por ejemplo solo tratan de verificar si existe o no un aplicativo, comúnmente un CGI, el cual posee una vulnerabilidad asociada. Es decir para el caso del CGI PHF, estos scanners solo realizan el requerimiento:

GET /cgi-bin/phf HTTP/1.0

o

HEAD /cgi-bin/phf HTTP/1.0

Dependiendo del código de respuesta generado por el servidor (200 Si existe) entonces se determina que el CGI PHF existe y que es vulnerable a un ataque que permite la ejecución de comandos.

Esta asociación de la existencia de un aplicativo con una vulnerabilidad que comúnmente se le asocia es la mayor causa de falsos positivos. Adicionalmente,

el realizar un requerimiento de este tipo corresponde a tráfico normal y por lo tanto nunca debe tratarse como una vulnerabilidad. Sin embargo si se desea filtrar este tipo de requerimientos es viable realizarlo adicionando estas cadenas al archivo de cadenas verificables strings.conf, aunque no es recomendable.

Es común que ocurra que al asociar una vulnerabilidad a la existencia de una aplicación, esta no posea dicha vulnerabilidad por que la aplicación corresponde a una versión que no es vulnerable, no se cumplen algunos requerimientos o se presentan algunas características atípicas, lo cual hace que la aplicación no sea vulnerable a algún problema determinado.

Afortunadamente existen scanners como nikto (www.cirt.net/code/nikto.shtml) y nessus (www.nessus.org) los cuales si realizan ataques para comprobar la existencia de una vulnerabilidad, aunque existen pruebas que corresponden a tráfico normal. Es por eso que estas 2 herramientas son las escogidas para generar los ataques en las pruebas. En conjunto estas herramientas están en capacidad de generar aproximadamente 3000 ataques.

La arquitectura para el desarrollo de las pruebas es la siguiente:

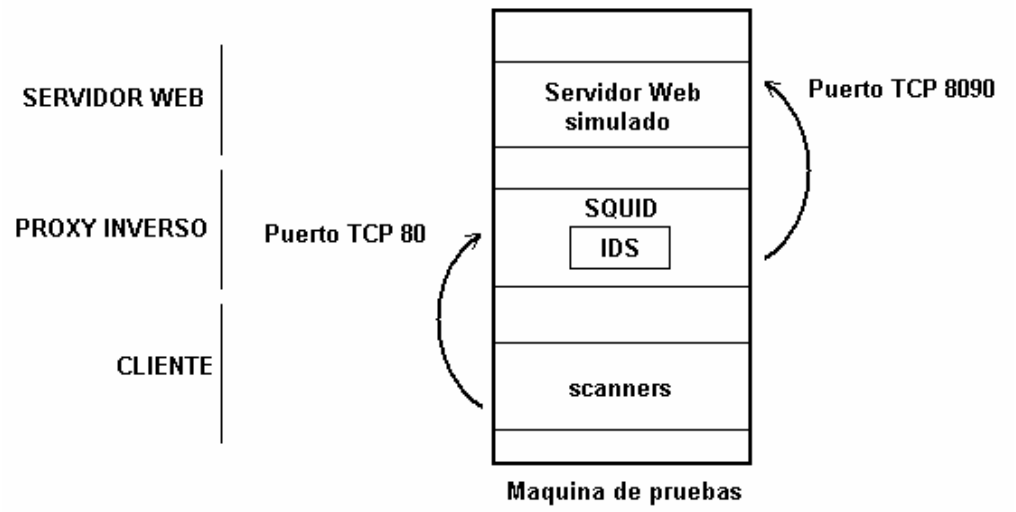

Figura 20. Arquitectura de pruebas

El servidor Web simulado corresponde a la utilidad netcat (utilidad simple para Unix que lee y escribe datos a través de conexiones de red usando los protocolos TCP o UDP) la cual permite simular un servidor TCP para capturar todos los requerimientos que lleguen a él y poder determinar el correcto funcionamiento de la infraestructura. Para tráfico normal se reconfigura SQUID para que apunte a un servidor Web externo (ej. fing.javeriana.edu.co) y el cliente convierte los requerimientos generados por un browser normal.

8.2 RESULTADOS

Las primeras pruebas que se desarrollaron fueron utilizando el scanner nikto. Este scanner realiza ataques reales aunque también, solo verifica la existencia de un archivo (ej. cgi) el cual es posiblemente vulnerable. Este tipo de verificación no es en realidad un ataque, se realiza por medio de un requerimiento normal y por lo tanto debe ser identificado como tráfico normal.

Adicionalmente se probó el mecanismo de la verificación de cadenas (ej. /etc/passwd) que aunque muchos de los ataques pueden ser detectados de esta manera, seria inconveniente basar las pruebas en este mecanismo teniendo en cuenta que un ataque puede estar dirigido a la obtención de otros objetivos y que el núcleo central del prototipo es la red neuronal. Por lo tanto es esta última la que es puesta a prueba como última barrera de defensa del sistema. Para ejecutar una prueba más fuerte, cada requerimiento es analizado y decodificado en su totalidad por la red neuronal por medio de la ventana corrediza que permite ponerla a prueba con patrones que nunca ha aprendido previamente. Así un requerimiento que contenga un ataque XSS (Cross Site Scripting), será analizado múltiples veces de la siguiente manera:

Requerimiento inicial:

[IDS] ANN cad: /%3c/title%3e%3cscript%3ealert(%22xss%22)%3c/script%3e

Decodificación:

[IDS] Decoded hex 3c to 60

[IDS] Decoded hex 3e to 62

[IDS] Decoded hex 3c to 60

[IDS] Decoded hex 3e to 62

[IDS] Decoded hex 22 to 34

[IDS] Decoded hex 22 to 34

[IDS] Decoded hex 3c to 60

[IDS] Decoded hex 3e to 62

Análisis:

SUBPatron normalizado:

/</@><@>@(

[IDS] ANN The output vector is :

0.000348 0.048149 0.992128 0.000040 0.033012

[ALERT!] CROSS SITE SCRIPT

SUBPatron normalizado:

</@><@>@("

[IDS] ANN The output vector is :

0.001678 0.018951 0.999886 0.000010 0.001635

[ALERT!] CROSS SITE SCRIPT

SUBPatron normalizado:

/@><@>@("@

[IDS] ANN The output vector is :

0.000467 0.026640 0.997739 0.000024 0.017545

[ALERT!] CROSS SITE SCRIPT

SUBPatron normalizado:

@><@>@("@"

[IDS] ANN The output vector is :

0.000309 0.012345 0.999641 0.000113 0.006848

[ALERT!] CROSS SITE SCRIPT

SUBPatron normalizado:

><@>@("@")

[IDS] ANN The output vector is :

0.005050 0.000114 0.999908 0.000167 0.000299

[ALERT!] CROSS SITE SCRIPT

SUBPatron normalizado:

<@>@("@")<

[IDS] ANN The output vector is :

0.002874 0.014233 0.999389 0.000006 0.000421

[ALERT!] CROSS SITE SCRIPT

SUBPatron normalizado:

@>@("@")</

[IDS] ANN The output vector is :

0.001885 0.000725 0.999670 0.000083 0.006378

[ALERT!] CROSS SITE SCRIPT

SUBPatron normalizado:

>@("@")</@

[IDS] ANN The output vector is :

0.009945 0.004433 0.998990 0.000041 0.000032

[ALERT!] CROSS SITE SCRIPT

SUBPatron normalizado:

@("@")</@>

[IDS] ANN The output vector is :

0.003481 0.004197 0.999716 0.000109 0.001700

[ALERT!] CROSS SITE SCRIPT

[TOTAL] 9

La siguiente tabla muestra los resultados obtenidos con *nikto*:

DETECTADOS	NO DETECTADOS

NORMAL	ATAQUES	CLASIFICACIÓN ERRADA	FALSOS POSITIVOS	FALSOS NEGATIVOS
323	846	183	49	28
1352			77	

Tabla 1. Prueba 1 - Resultados Prototipo / Nikto

Total pruebas = 1429

El sistema analizó 1429 requerimientos de los cuales 1057 correspondían a verdaderos ataques. De dichos ataques 846 fueron detectados e identificados plenamente, aunque otros 183 fueron detectados pero clasificados erróneamente.

Analizando uno a uno los resultados se pudo determinar que la totalidad de la clasificación errónea y la gran mayoría de falsos positivos y negativos se debe a que existían patrones básicos que no estaban incluidos en el entrenamiento tales como:

PATH;./../../@/
NORMAL;@=@_@@@@@@

Para determinar los porcentajes de efectividad del sistema, hay que tener en cuenta que la clasificación errónea corresponde a una correcta detección de un ataque el cual fue clasificado erróneamente, debido a que la red neuronal no estaba entrenada para manejar el patrón determinado. Teniendo en cuenta los valores anteriores se determinó:

(%) Porcentaje de Detección: **94,6%**
(%) Porcentaje de Identificación: **81,8%**

NOTA:

Total Detectados = Normal + Ataques + C. Errada

*% de Detección = (Total Detectados * 100) / Total pruebas*

Total Identificados = Normal + Ataques

*% de Identificación = (Total Identificados * 100) / Total pruebas*

Para mejorar el porcentaje de detección y en particular el de identificación, se introdujeron nuevos patrones y posterior al reentrenamiento de la red neuronal, incluyendo 10 patrones básicos nuevos, se realizó nuevamente la prueba con los siguientes resultados:

DETECTADOS			NO DETECTADOS	
NORMAL	**ATAQUES**	**CLASIFICACIÓN ERRADA**	**FALSOS POSITIVOS**	**FALSOS NEGATIVOS**
341	886	154	31	17
1381			**48**	

Tabla 2. Prueba 2 - Resultados Prototipo / Nikto

Total pruebas = 1429

(%) Porcentaje de Detección: **96,6%**

(%) Porcentaje de Identificación: **85,8%**

Si se analizan los anteriores resultados, al aumentar la efectividad del sistema, se puede observar que una disminución en falsos positivos, incrementa el numero de requerimientos normales detectados y una disminución de falsos negativos aumenta el numero de ataques clasificados correcta o

incorrectamente. Al realizar las pruebas con el scanner *nessus* se obtuvieron los siguientes resultados:

DETECTADOS			NO DETECTADOS	
NORMAL	**ATAQUES**	**CLASIFICACIÓN ERRADA**	**FALSOS POSITIVOS**	**FALSOS NEGATIVOS**
525	7004	2639	958	368
10168			**1326**	

Tabla 3. Prueba 1 - Resultados Prototipo / Nessus

Total pruebas = 11494

(%) Porcentaje de Detección: **88,4%**

(%) Porcentaje de Identificación: **65,5%**

Al analizar cada una de las respuestas se pudo evidenciar nuevamente y en número superior la existencia de patrones que no existen en el set de entrenamiento y por lo tanto disminuye la efectividad del sistema. En el momento de realizar nuevamente esta prueba adicionando los patrones faltantes, 200 nuevos patrones aproximadamente, se obtuvieron los siguientes datos.

DETECTADOS			NO DETECTADOS	
NORMAL	**ATAQUES**	**CLASIFICACIÓN ERRADA**	**FALSOS POSITIVOS**	**FALSOS NEGATIVOS**
816	8653	439	496	98
9908			**594**	

Tabla 4. Prueba 2 - Resultados Prototipo / Nessus

Total pruebas = 10502

(%) Porcentaje de Detección: **94,3%**

(%) Porcentaje de Identificación: **90,1%**

En la actualidad existen 794 patrones en el set de entrenamiento para lo cual cabe resaltar que este set es de dimensiones mínimas comparadas con los altos porcentajes de detección en los resultados actuales, esto determina la viabilidad del sistema para detectar nuevos ataques e identificarlos, aunque esta última tarea depende de la existencia de un set de entrenamiento más numeroso que el actual para mejorar significativamente el porcentaje de identificación. Adicionalmente las pruebas realizadas se basan en el peor escenario donde cada requerimiento es analizado por la red neuronal, después de pasar libremente niveles superiores como el filtrado de características. También es necesario mencionar que la configuración actual de la ventana corrediza que fracciona los requerimientos para entregarlos a la red neuronal, lo hace con un corrimiento de longitud 1, lo cual incrementó el tamaño del set de entrenamiento (794 patrones) debido a que aumentan las posibles combinaciones de un patrón determinado mientras la ventana se desliza por la totalidad del requerimiento analizado.

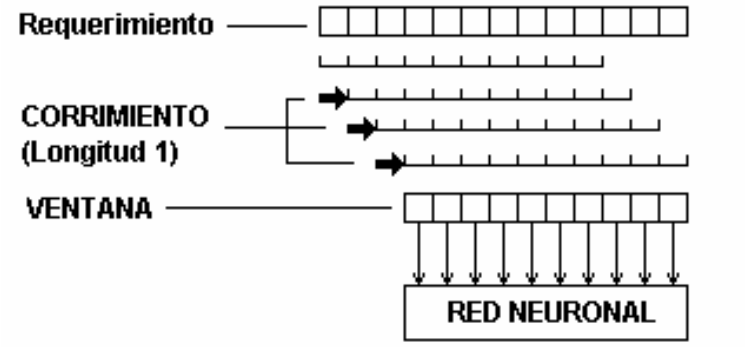

Figura 21. Deslizamiento de Ventana

Adicionalmente se pudo establecer que todo ataque detectado generó en la salida un valor superior a 0.9, lo cual indica que la red neuronal es poco sensible a variaciones en sus entradas. Esto muestra que para la red neuronal la forma como se normalizan los datos y su clasificación facilita su detección,

debido a que se obtienen respuestas bien definidas por parte de la red neuronal. La limitación más evidente de la red neuronal en el sistema corresponde a ataques que son reproducidos utilizando requerimientos totalmente normales. Un ejemplo básico de esta limitación es la vulnerabilidad existente en algunos programas de compra para comercio electrónico donde el archivo de tarjetas de crédito es almacenado en un directorio que puede ser accedido directamente por medio de un requerimiento normal (ej. Mountain-net WebCart v.1.0 - http://www.securityfocus.com/bid/2281/discussion/) . El exploit particular para el anterior ejemplo es http://host_vulnerable/orders/checks.txt el cual corresponde a un requerimiento totalmente normal. Si se analiza este tipo de problemas la gran mayoría corresponde a problemas del diseño de la arquitectura de la aplicación. Donde se maneja un diseño en el cual los archivos críticos de la aplicación, configuración, administración y datos, se encuentran almacenados en directorios dentro del la raiz Web del servidor y por lo tanto pueden ser accedidos directamente sin ningún problema. Una variación de este problema es común en aplicaciones donde se debe manejar sesiones por usuario pero no se implementa un identificador de sesión. Por lo general dicha sesión es autenticada pero al no manejar un identificador de sesión de algún tipo es viable apuntar, por medio de un requerimiento normal, hacia una fase posterior a la autenticación eliminando este mecanismo de seguridad. Lo importante es recalcar que estas consideraciones corresponden al diseño de la aplicación y no a la implementación de las mismas. Así, este tipo de problemas debe ser manejado por el filtro de características donde la presencia de la cadena del requerimiento normal, determina el ataque.

Para poder determinar la calidad de los resultados se realizaron las mismas pruebas al NIDS Snort versión 2.0 (http://www.snort.org) para poder así comparar los porcentajes de detección resultantes. La decisión sobre el uso de Snort como IDS comparativo, se basa en que este programa es de libre distribución, de código abierto y de amplia difusión. Cabe mencionar que aunque Snort esta diseñado para detectar ataques en otros protocolos adicionales a

HTTP, los ataques solo fueron hechos utilizando las mismas herramientas (nikto v1.23, nessus v2.0.5) en las mismas condiciones en las que se generaron las pruebas al prototipo para garantizar resultados coherentes y reales. Es importante recalcar que no se hace una comparación de los porcentajes de identificación debido a que cada IDS maneja su propia clasificación. Los resultados obtenidos para Snort utilizando *nessus* fueron los siguientes:

ATAQUES	
DETECTADOS	**NO DETECTADOS**
303	**207**

Tabla 5. Resultados Snort / Nessus

Total ataques = 510

(%) Porcentaje de Detección: **59,4%**

Los resultados obtenidos para Snort utilizando *nikto* fueron:

ATAQUES	
DETECTADOS	**NO DETECTADOS**
1256	**420**

Tabla 6. Resultados Snort / Nikto

Total ataques = 1676

(%) Porcentaje de Detección: **74,9%**

Los porcentajes de detección resultantes demuestran que el prototipo desarrollado es un 23,1% más efectivo que Snort.

IDS	NIKTO		NESSUS		% DETECCIÓN
	DETECTADOS	NO DETECTADOS	DETECTADOS	NO DETECTADOS	
SNORT	1256	420	303	207	**71,3**
PROTOTIPO	1381	48	9908	594	**94,4**

Tabla 7. Porcentajes de Detección Totales - Prototipo vs Snort

Para entender la relación entre el número de pruebas analizadas por el prototipo y el número de ataques analizados por Snort, es necesario tener en cuenta que el modo como se analizan los datos en cada uno es totalmente diferente pues el prototipo realiza una mayor cantidad de análisis debido a la ventana corrediza que se desliza por cada ataque, por lo tanto en un requerimiento existen múltiples análisis según su composición y longitud, mientras que los IDS convencionales, como Snort, solo realizan un análisis por ataque.

Para ejemplificar mejor la anterior relación, tenemos la siguiente cadena de ataque:

/directorio1/directorio2/vulnerable?parametro1=valor&parametro2=../archivo

En Snort esta cadena es analizada una sola vez en busca de la presencia de la firma, mientras que el prototipo realiza su normalización para entregarla a la red neuronal de forma segmentada a través de la ventana corrediza (corrimiento=1, longitud=10) así:

Cadena normalizada: /@/@/@?@=@&@=../@

Análisis 1 - Segmento 1:	/@/@/@?@=@
Análisis 2 - Segmento 2:	@/@/@?@=@&
Análisis 3 - Segmento 3:	/@/@?@=@&@
Análisis 4 - Segmento 4:	@/@?@=@&@=
Análisis 5 - Segmento 5:	/@?@=@&@=.
Análisis 6 - Segmento 6:	@?@=@&@=..
Análisis 7 - Segmento 7:	?@=@&@=../

Análisis 8 - Segmento 8: @=@&@=../@

Total de análisis. 8

Por lo tanto, aunque se pueden comparar directamente los porcentajes de detección, pues este es el reflejo del objetivo final de este tipo de aplicaciones, no es posible comparar otro tipo de datos que son dependientes de la forma como cada IDS trabaja.

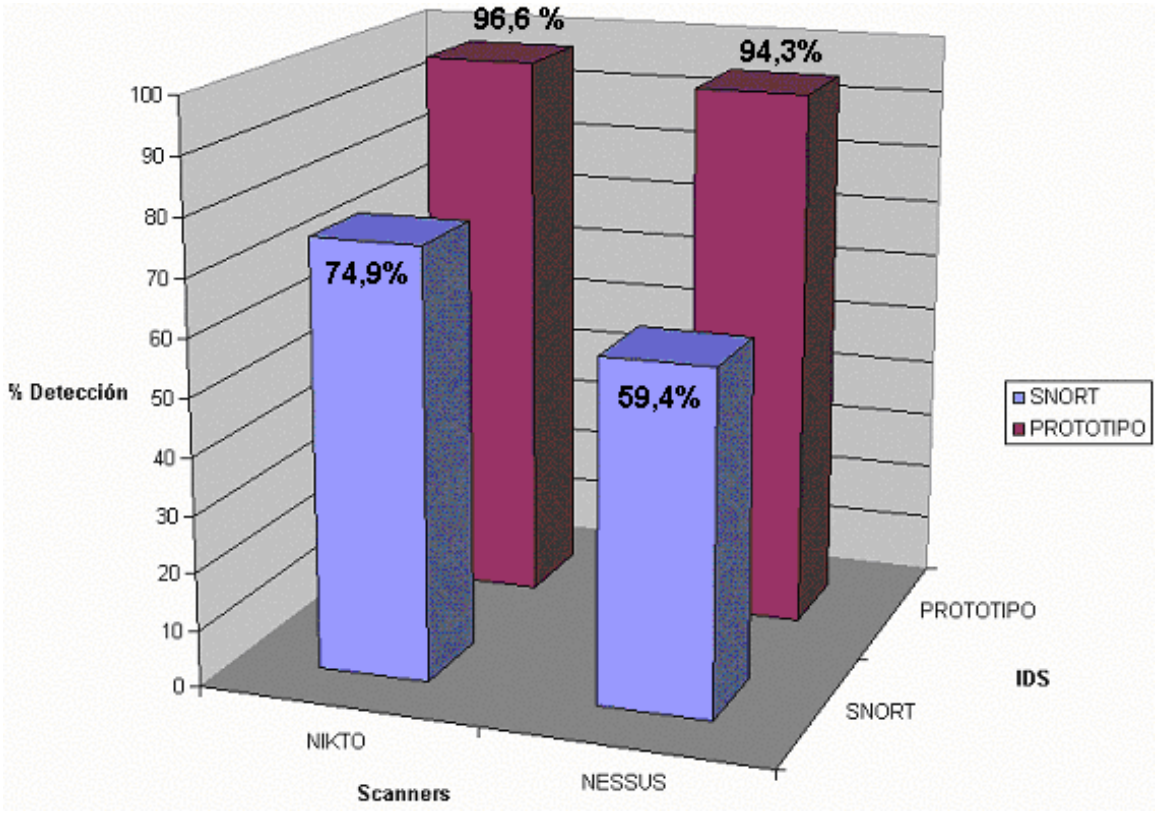

Figura 22. Porcentajes de Detección Individuales - Prototipo vs Snort

Los resultados demuestran que es viable implementar un sistema de detección de intrusos que elimina las limitaciones de los IDS actuales por medio de un

diseño simple multi-nivel balanceado, una arquitectura de red acorde a las necesidades reales y unas tecnologías adaptivas, como las redes neuronales, que replantean la forma como comúnmente se han clasificado, detectado e identificado las vulnerabilidades conocidas y por conocer.

9. CONCLUSIONES

Durante el desarrollo de este trabajo se han generado múltiples conclusiones que son parte fundamental de los objetivos de este proyecto, pero también han servido como soporte para la toma de decisiones en el desarrollo de este trabajo. El siguiente es un compendio de las conclusiones más importantes.

- La clasificación de vulnerabilidades es viable, si dicha clasificación esta basada en las causas y no en los efectos de las mismas.

- La detección de ataques conocidos y principalmente no conocidos en el protocolo HTTP debe estar basado en el desacoplamiento de la vulnerabilidad con el recurso que la posee, lo cual esta íntimamente ligado con la forma como se presentan los datos hacia el mecanismo de análisis y detección, principalmente en el uso de redes neuronales.

- La gran mayoría de las limitaciones en los IDS actuales provienen de la arquitectura en el nivel OSI en que trabajan, así a mayor nivel, menores serán las limitaciones dependientes de la arquitectura.

- La detección de ataques basados en el desbordamiento de pila (stack overflows) por medio de la búsqueda de patrones en su shellcode asociado es sumamente ineficiente y de fácil subversión debido a su carácter dinámico y polimórfico, para lo cual es mejor enfocarse en la protección por medio del refuerzo de los límites de desbordamiento.

- El carácter polimórfico de los shellcodes hacen que su detección por medio de redes neuronales sea inadecuado, pues conllevan las mismas limitaciones de los antivirus tradicionales al tratar de detectar virus polimórficos.

- Para la determinación de los límites de desbordamiento a reforzar es necesario tener en cuenta que el uso normal de una aplicación esta inscrito dentro de los límites de los requerimientos normales (tráfico normal) los cuales siempre están circunscritos dentro de los límites máximos que evitan ataques que aprovechan el desbordamiento de algún segmento en la pila.

- El uso de redes neuronales de tipo Elman para el análisis y detección de intrusos a nivel de protocolo HTTP es viable por medio del procesamiento adecuado de las entradas y una clasificación de ataques clara y concisa la cual permita diferenciar lo normal de lo anormal.

- El uso de un Proxy inverso como punto central para la protección de servidores web solo debe ser implementado si el Proxy realiza un análisis completo de la totalidad de los datos incluidos en un requerimiento para la detección de ataques. Adicionalmente esto permite pasar de un sistema de detección (IDS) a un sistema de prevención (IPS).

- La arquitectura básica del Proxy SQUID, por medio de su interfaze redirectora no permite ser usado como mecanismo de filtrado de requerimientos y menos como IDS. Para esto se requiere realizar modificaciones internas que provean el nivel de información adecuado para la toma de decisiones.

- Los límites establecidos en los firewalls de aplicación comerciales solo protegen a los servidores Web pero no a las aplicaciones que soportan.

- Un sistema IDS eficiente, como mínimo debe poseer un nivel de decodificación inicial y un nivel de detección posterior. Así entre las grandes

limitaciones de los NIDS convencionales se encuentra la imposibilidad del desarrollo adecuado de un nivel de decodificación inicial para el tráfico encriptado.

- El balanceo de tareas de los niveles establecidos en el IDS depende directamente de las limitaciones de las tecnologías implementadas en cada uno de ellos, del funcionamiento básico de los ataques a detectar y de la disminución en la complejidad de cada nivel.

- Es viable implementar un sistema de detección de intrusos que elimina las limitaciones de los IDS actuales por medio de un diseño simple multi-nivel balanceado, una arquitectura de red acorde a las necesidades reales y tecnologías adaptivas, como las redes neuronales, que replantean la forma como comúnmente se han clasificado, detectado e identificado las vulnerabilidades conocidas y por conocer.

10. CONTINUIDAD Y RECOMENDACIONES

El prototipo pretende ser un punto de partida para nuevas investigaciones y desarrollos en el área de la seguridad informática, redes neuronales y en especial, el diseño e implementación de IDS que estén por fuera del modelo actual. La siguiente lista presenta algunos posibles caminos para el mejoramiento del prototipo y el desarrollo de nuevas investigaciones a partir de el.

Para el mejoramiento del prototipo o nuevas investigaciones:

- Aplicación del Q-Test (Prueba estadística para determinar si un punto en un conjunto de datos es significativamente diferente para ser descartado) a los datos recolectados en limits.log para determinar la detección de requerimientos anómalos.

- Generar los rangos mínimos y máximos de parámetros numéricos en los requerimientos como medio de protección contra ataques del tipo “integer overflows”.

- Desarrollo de un Proxy inverso para la detección de intrusos a nivel de protocolo HTTP u otros protocolos.

- Modificación de la arquitectura de SQUID para adecuarla a un sistema IDS.

- Analizar la posibilidad de definir el enfoque presentado en otros protocolos a nivel de aplicación.

- Análisis comparativo entre diferentes tipos de redes neuronales para la detección de intrusos a nivel de aplicación.

- Para el mejoramiento de la aplicación es posible el desarrollo de interfaces gráficas que automaticen los procesos manuales actuales y generen estadísticas analizables gráficamente por el administrador del IDS.

- Analizar la posibilidad de generar tecnologías de sandboxing para la detección de shellcodes.

- Determinar el conjunto de ataques para los cuales la red neuronal Elman desarrollada infiere mejor.

- Desarrollar nuevas técnicas de presentación de datos para redes neuronales encaminadas a la detección de intrusos.

GLOSARIO

Exploit:

Mecanismo que aprovecha una o más vulnerabilidades existentes para romper la seguridad de uno o más sistemas vulnerables.

Honeypot:

Recurso de seguridad cuyo valor reside en ser sondeado, atacado o comprometido por atacantes. No tiene valor productivo, cualquier cosa que se envíe a éste recurso (Ej. servidor) es con toda seguridad un sondeo, ataque o compromiso.

Format bug:

Vulnerabilidad que generalmente ocurre por fallas en la implementación de funciones que manejan cadenas de formato. Las cuales pueden ser manipuladas para modificar el contenido de la pila y modificar el comportamiento normal de un proceso activo.

Buffer overflow:

Vulnerabilidad que ocurre cuando se almacenan mas datos de los que el buffer definido en la pila pude almacenar, desbordando sus límites y sobrescribiendo datos adyacentes, corrompiendo o modificando su contenido para modificar el comportamiento normal del proceso.

Off-by-one overflow:

Caso particular del buffer overflow cuando en un buffer de tamaño i y capacidad N, solo es posible generar su desbordamiento cuando se introducen datos de longitud N.

Heap overflow:

Vulnerabilidad asociada a los buffer overflows pero manipulando los datos almacenados en el Heap o en el segmento BSS.

SQL injection:

Vulnerabilidad que ocurre por fallas en el control de los datos de entrada asociados a una consulta SQL, la cual puede ser modificada incluyendo caracteres especiales que al ser ejecutada modifican su comportamiento normal.

Falso Negativo:

En los IDS es definido como el fallo al no detectar un ataque presente.

Falso Positivo:

En los IDS es definido como el fallo al detectar un ataque inexistente.

Negación del Servicio / Denial of Service (DoS):

Ataque específicamente diseñado para prevenir el normal funcionamiento de uno o mas sistemas. Negando el acceso y/o uso normal de los servicios prestados por los sistemas a sus usuarios.

BIBLIOGRAFÍA

[0] Torres, Efraín. “Sistema inmunológico para la detección de intrusos a nivel de protocolo HTTP”. Propuesta de proyecto de grado. Pontificia Universidad Javeriana, Bogotá, 2002.

[1] A. Hofmeyr, S. Forrest ," Immunity by Design: An Artificial Immune System", University of New Mexico, Albuquerque, 1997

[2] D'haeseleer, S. Forrest, y P. Helman. "An immunological approach to change detection: Algorithms, analysis and implications". In Proceedings of the 1996 IEEE Symposium on Research in Security and Privacy, Los Alamitos, CA, 1996. IEEE Com-puter Society Press.

[3] S. A. Hofmeyr, S. Forrest, y A. Somayaji. "Intrusion detection using sequences of system calls." Journal of Computer Security, 6:151-180, 1998.

[4] C. Warrender, S. Forrest y B. Pearlmutter "Detecting Intrusions Using System Calls: Alternative Data Models", University of New Mexico, Albuquerque, 1999

[5] S21SEC, http://www.s21sec.com

[6] Network Working Group, “Hypertext Transfer Protocol - HTTP/1.1”, RFC 2616. http://www.w3.org/Protocols/rfc2616/rfc2616.html

[7] Ghosh K. Anup, Schwartzbard A. Schatz M. "Learning Program Behavior Profiles for Intrusion Detection" Reliable Software Technologies Corporation. 1999.

[8] Lippman, R. Conningham, R. "Improving Intrusion Detection Performance Using Keyword Selection and Neural Networks". MIT Lincoln Laboratory, 2000

[9] Draelos, T. Collins, Michael. "Experiments on Adaptive Techniques for Host-Based Intrusion Detection" Sandia National Laboratory, SAND2001-3065, 2001

[10] Anderson D. McNeill G. "Artificial Networks Technology", Kaman Sciences Corporation. 1992

[11] Elman, Jeffrey. "Finding Structure in Time". Cognitive Science, 14. 179-211 1990.

[12] LoWNOISE,“Wmap v1.4 web scanner”, http://pwp.007mundo.com/etorres1/

[13] S. A. Hofmeyr, S. Forrest, y A. Somayaji. "Lightweight Intrusion Detection for Networked Operating Systems." University of New Mexico, Albuquerque, 1998

[14] S. A. Hofmeyr, S. Forrest, y A. Somayaji."Principles of a Computer Immune System" University of New Mexico, Albuquerque, 1998

[15] Art Stricek, “A Reverse Proxy Is A Proxy By Any Other Name”, 2002, http://www.sans.org/rr/web/reverse_proxy.php

[16] SQUID, http:// www.squid-cache.org

[17] SQUID FAQ, http://www.squid-cache.org/Doc/FAQ/FAQ-15.html

[18] Jeanne: Modified Reverse Proxy, Dartmount College, Institute for Security Technology Studies.
http://www.ists.dartmouth.edu/IRIA/projects/d_jeanne.htm

[19] Dorene L. Kewley, John Lowry, “Observations on the effects of defense in depth on adversary behavior in cyber warfare”, Proceedings of the 2001 IEEE, Workshop on Information Assurance and Security, United States Military Academy, West Point, NY, Junio, 2001.

[20] FozZy, « Advanced Shellcodes”, Defcon X, Las Vegas,USA, 2002.

[21] Fermín J. Serna, “ Polymorphic Shellcodes vs. Application IDSs, Next Generation Security Technologies, http://www.ngsec.com

[22] Eeye SECUREIIS v 1.2.1 USER MANUAL http://www.eeye.com

[23] Bugtraq, SecurityFocus, http://www.securityfocus.com

[24] Kumar, S. and Spafford, E., "A Pattern Matching Model for Misuse Intrusion Detection," Proceedings of the Seventeenth National Computer Security Conference, pp. 11--21 (Oct. 1994).
http://citeseer.nj.nec.com/kumar94pattern.html

[25] Husmousa, “Neural Net based Host/Application Anomaly detection systems” http://www.securityfocus.com/archive/96/257986/2002-02-17/2002-02-23/1

[26] Farkas, Jennifer. “Document Classification and Recurrent Neural Networks”, Industry Canada, Centre for Information Technology Innovation (CITI).

[27] Unicode IIS. Bugtraq. http://www.securityfocus.com

[28] Clowes, Shaun. “A Study In Scarlet, Exploiting Common Vulnerabilities in PHP Applications”, SecureReality. http://www.secureReality.com.au/archives/studyinscarlet.txt

[29] Rain Forest Puppy, “A look at whisker's anti-IDS tactics”, http://www.wiretrip.net/rfp/pages/whitepapers/whiskerids.html

[30] Gary McGraw, John Viega. “Make your software behave: Learning the basics of buffer overflows- Get reacquainted with the single biggest threat to software security” Reliable Software Technologies, March 1, 2000.

[31] Schoenefeld, Marc. “Denial-Of-Service holes in JDK 1.4.1_01” Security Alert. http://www.illegalaccess.de/

[32] Bugtraq, “PHF Remote Command Execution Vulnerability”, 1996 http://www.securityfocus.com/bid/629

[33] Aleph One, “Smashing the Stack for fun and profit”, Phrack 49. 1996

[34] Klog, “The Frame Pointer Overwrite”, Phrack 55, 1999.

[35] Matt Conover, w00w00 Security Team. “w00w00 on Heap Overflows”, 1999.
http://www.w00w00.org/files/articles/heaptut.txt

[36] Blexim, “Basic Integer Overflows”, Phrack 60. 2002.

[37] Scut, team teso. “Exploiting Format String Vulnerabilities”, 2001.

[38] SpinNews, “Code Red Update”.
http://www.photospin.com/SpinNews/archive/August/bugs.html

[39] Core Security, “Vulnerabilities in your code – Advanced Buffer Overflows” v1.2 Octubre 31, 2002. http://www.core-sec.com

[40] Matias Sedalo, ·”JempiScodes - Polymorphic Shellcode Generator” 2003
http://www.shellcode.com.ar/

[41] K2, ADM Mutate.
http://packetstormsecurity.nl/defcon10/dc10-fozy-shellcodes/dc10-fozy-WritingShellcodes/tools/

[42] Julien Olivain, Epitech IV. Le polymorphisme et le camouflage des shellcodes . 2002.

[43] Mitchell, John. ‘’ Virus protection and intrusion detection’’ .
http://crypto.stanford.edu/cs155/lecture13.pdf

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

[45] J.P Anderson. “Computer Security Threat Monitoring and Surveillance. Technical report”, James P Anderson Co., Fort Washington, Pennsylvania, April 1980.

[46] Patrick Gray. “This little piggie guards the market”, The Age, Australia, 2002 http://www.theage.com.au/articles/2002/09/22/1032055006051.html

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

ANEXO 1: MANUAL DE USUARIO

El siguiente documento describe el uso y operación del IDS desarrollado. Como se podrá observar, el IDS ha sido desarrollado para disminuir al máximo la necesidad de modificaciones y acciones durante su ejecución por parte del usuario administrador del sistema. Por lo tanto la complejidad del sistema radica en su instalación y configuración inicial, no en su uso normal.

NOTA: El software desarrollado corresponde a un prototipo funcional, por lo tanto debe ser tratado como tal y no como un producto final desarrollado para ambientes reales en producción. Adicionalmente, se realizaron adecuaciones a la arquitectura de SQUID, las cuales no deben ser tomadas como un mejoramiento a dicha arquitectura, deben ser tomadas solo como una adecuación teniendo en cuenta que la arquitectura normal de este Proxy no provee la funcionalidad necesaria para un sistema de detección de intrusos a nivel de aplicación para protocolo HTTP.

Utilidad idsconfig.

El programa idsconfig permite generar automáticamente un archivo de configuración (ids.conf) básico adecuado para el ambiente actual, basado en los datos provenientes del archivo de límites limits.log. Adicionalmente permite generar archivos graficables con la herramienta Xplot. Estos archivos al ser graficados permiten observar el comportamiento de los tamaños de cada parte de los requerimientos almacenados, con el motivo de determinar y analizar los límites adecuados configurados en el archivo ids.conf y ajustar dicho archivo a las necesidades reales del servidor Web protegido, en conjunto con sus aplicaciones.

Para obtener el conjunto de opciones del programa idsconfig ejecute:

Idsconfig -h

Para graficar límites almacenados en el archivo limits.conf:

Idsconfig <flag_límites_deseados> > <archivo_xplot>
Xplot <archivo_xplot>

Para generar una configuración base según los datos almacenados en limits.log:

Idsconfig –f > <archivo_configuracion_ids>

Para observar los límites actualmente almacenados, ejecute:

Idsconfig –v

El uso adecuado de este programa corresponde a generar una base para la configuración del IDS y ajustarla manualmente según el análisis de las graficas. Por lo general, si la configuración del IDS es realizada en un ambiente controlado, el ajuste manual de la configuración se convierte en un paso innecesario.

Generación de alarmas

Todo ataque detectado será almacenado en el archivo ids.log, dicho archivo posee el siguiente formato:

HORA_FECHA IP_ORIGEN DESCRIPCION_ATAQUE

El tratamiento que se hace de ellas es responsabilidad del redirector asociado y es decisión del usuario determinar la acción requerida en el caso de un ataque.

Ids.conf

Las siguientes opciones son las que comúnmente pueden ser manipuladas durante la ejecución del IDS: Los valores de estas variables corresponden a valores 0 y 1 donde 1 es verdadero y 0 falso.

CHECK_STRINGS

Esta opción permite verificar la existencia de cadenas definidas en el archivo strings.conf. Estas cadenas corresponden a recursos críticos del sistema protegido, cuyo intento de acceso, representado por la existencia de la cadena en el requerimiento, significa un posible intento de ataque por lo cual debe ser detectado. Se recomienda mantener el listado de dichas cadenas al mínimo y ajustarlas según el sistema protegido y sus aplicaciones.

CHECK_LIMITS

Esta opción permite analizar los límites de cada parte de un requerimiento, según las especificaciones dadas en el archivo de configuración ids.conf.

CHECK_NN

Esta opción es la encargada de habilitar o deshabilitar el análisis de los requerimientos por parte de la red neuronal desarrollada en el IDS. Este análisis

está basado en determinar la existencia de patrones que correspondan a ciertos tipos de ataques conocidos y no conocidos en el protocolo http.

GEN_LIMITS

Esta opción permite generar los límites de los requerimientos normales, los cuales son almacenados en el archivo limits.log para su posterior análisis. Se recomienda deshabilitar esta opción después de poseer una configuración apta, valida y perfeccionada para el servidor protegido.

DETAIL

Esta opción habilita el almacenamiento de la totalidad del requerimiento si en él se determina un ataque.

OVERRIDE_GENLIM

Esta opción permite almacenar los límites de los requerimientos aunque exista un ataque detectado en ellos. Esta opción se utiliza para realizar análisis de los datos pero no se recomienda generar configuraciones a partir de ellos. Se recomienda mantener deshabilitada esta opción.

ANEXO 2: MANUAL DE INSTALACIÓN Y CONFIGURACIÓN

El siguiente documento describe el procedimiento detallado para la instalación del IDS y su configuración total para la protección de un servidor Web determinado.

NOTA: El software desarrollado corresponde a un prototipo funcional, por lo tanto debe ser tratado como tal y no como un producto final desarrollado para ambientes reales en producción. Adicionalmente, se realizaron adecuaciones a la arquitectura de SQUID, las cuales no deben ser tomadas como un mejoramiento a dicha arquitectura, deben ser tomadas solo como una adecuación teniendo en cuenta que la arquitectura normal de este Proxy no provee la funcionalidad necesaria para un sistema de detección de intrusos a nivel de aplicación para protocolo HTTP.

Requerimientos

Los requerimientos a nivel de hardware son bastante simples. Solo se requiere un PC con los siguientes requerimientos mínimos:

Hardware

- Procesador: Pentium II (200 Mhz)
- Memoria: 128M RAM
- Espacio D.D: 30M Libres en su mayoría para logs.

- Tarjeta de Red

Software

- Sistema Operativo: Linux (Mandrake 9 / Redhat 8.2)
- Compilador C (Gcc)

Obtener SQUID e instalarlo en conjunto con el IDS

Obtenga el archivo squid-2.5.STABLE1-src.tar.gz. Para los detalles de instalación refiérase a la documentación detallada de SQUID en el sitio www.squid-cache.org. Posterior a la obtención del código fuente realice el siguiente procedimiento para descomprimir el código fuente:

```
tar -xvzf squid-2.5.STABLE1-src.tar.gz
```

Ubique el archivo que contiene el código fuente del IDS (ids.tar.gz) y descomprímalo así:

```
tar –xvzf ids.tar.gz
```

Posterior a su descompresión vaya al directorio ids/src/code y copie la totalidad de su contenido al directorio squid-2.5.STABLE1/src así:

```
cd ids/src/code
cp *  squid-2.5.STABLE1/src
```

Estando adentro del directorio de src de SQUID siga el procedimiento normal para la instalación de SQUID.

```
./configure
make
(como usuario root)
make install
```

Configuración de Squid

SQUID normalmente funciona como un proxy cache. Para configurarlo como Proxy inverso, edite el archivo:

```
/etc/squid.conf
```

Así:

```
# squid.conf – configuración básica para Proxy inverso
# Hacer que el proxy inverso escuche por el puerto 80 como cualquier Web
# server normal.
#
http_port 80
#El nombre visible del host a proteger
visible_hostname www.my-domain.com
# dirección IP y puerto real del webserver detrás del Proxy inverso.
httpd_accel_host 10.20.30.4
httpd_accel_port 80
# Apagar el proxy original.
httpd_accel_with_proxy off
```

```
# Configurar cache valido por 30 segs.
refresh_pattern . 0 0% 30
# Listas de control de acceso

.. (ver documentacion de SQUID)

# PATH del redirector asociado.
redirect_program /ids/bin/basico
redirector_bypass off
redirect_children 3
...
```

Compilación del redirector asociado

En la carpeta /ids/src compile basico.c y copie al directorio /ids/bin, asi:

```
gcc  -o basico basico.c
cp basico /ids/bin
```

Revise los permisos de ejecución de este archivo para que pueda ser ejecutado por squid.

Adicionalmente compile el programa idsconfig y copuelo también a /ids/bin.

```
gcc –o idsconfig idsconfig.c
cp idsconfig /ids/bin
```

Configuración inicial del IDS

El archivo /ids/etc/ids.conf contiene la configuración básica del IDS. Este archivo debe ser modificado para ser adecuado a las características normales de funcionamiento del servidor Web protegido. Inicialmente la configuración del IDS debe ser lo mas genérica posible para luego ser ajustada por medio de herramientas como idsconfig (ver. Manual de usuario).

Defina la ubicación de los siguientes archivos:

LOG_FILE= /usr/local/squid/ids/log/ids.log (Archivo de Log del IDS)
LIMITS_FILE= /usr/local/squid/ids/log/limits.log (Archivo de límites)
STRINGS_FILE= /usr/local/squid/ids/etc/strings.conf (Archivo de cadenas)

Compruebe que las siguientes variables esten configuradas inicialmente de la siguiente manera:

(Verificar la presencia de cadenas extrañas.)
CHECK_STRINGS= 1
(Verificar los límites de tamaño de cada requerimiento)
CHECK_LIMITS= 1
(Verificar utilizando la red neuronal)
CHECK_NN= 1
(Generar límites)
GEN_LIMITS= 1
(Almacenar el requerimiento si en el se encuentra un ataque)
DETAIL= 1
(Generar límites aunque exista un ataque)
OVERRIDE_GENLIM= 0

Posterior a esta verificación, configure los límites a valores genéricos así:

GENERIC_REQUEST_SIZE: 0,1024
METHOD_SIZE: 0,10
URI_GET_SIZE: 0,1024
URI_POST_SIZE: 0,1024
HEADERS_SIZE: 0,1024
HEADER_Connection_SIZE: 0,250
HEADER_User-Agent_SIZE: 0,250
HEADER_Host_SIZE: 0,250
BODY_SIZE: 0,1024
PARAMETERS: 0,1024

En estos momentos todo esta listo para iniciar squid y automáticamente se iniciara el IDS con su redirector asociado. Ejecute squid así:

cd /usr/local/squid/bin/
./squid

Desde este momento cada requerimiento será analizado y almacenados sus límites, si es normal, en el archivo limits.log. Por lo tanto para un ajuste rápido de los límites genéricos anteriormente configurados, trate de generar el mayor tráfico posible a su servidor Web, tratando de cubrir todos los casos de uso que provee su servidor Web y aplicaciones instaladas sobre él. Se recomienda que este proceso sea realizado en un ambiente controlado para evitar que los datos sean alterados con valores críticos.

Al poseer un número considerable de datos almacenados en limits.log, utilice el programa idsconfig para analizar dichos datos gráficamente y generar un archivo de configuración (ids.conf) adecuado para su ambiente. (ver Manual de usuario).

ANEXO 3: PATRONES BÁSICOS DE ENTRENAMIENTO RED NEURONAL

El siguiente anexo presenta un listado de los patrones básicos recolectados para el entrenamiento de la red neuronal Elman. Los cuales fueron obtenidos inicialmente al analizar vulnerabilidades publicadas en listas de correo como Bugtraq (http://www.securityfocus.com).

NOTA: Formato *<Tipo>;<patrón>* Num. Patrones 794

INJECTION;&@=";@ ";
INJECTION;&@=;@ /@/@
INJECTION;'(@
INJECTION;.@/@=;@ @.
INJECTION;.bat?&@
INJECTION;.bat|@ @:\
INJECTION;.EXE? -@ @
INJECTION;.exe|@
INJECTION;/@&@_@= ;@
INJECTION;/@.@/@=;@
INJECTION;/@.bat|@ @
INJECTION;/@/@ /@/@|
INJECTION;/@/@.@/@=;
INJECTION;/@/@.bat|@
INJECTION;/@/@|&@_@
INJECTION;/@/@|&@_@=
INJECTION;/@;@ /@/@|
INJECTION;"; @ `@ `
INJECTION;'; @ `@ `;
INJECTION;;&@=/@/@ /
INJECTION;;@ /@/@
INJECTION;;@ /@/@|&@
INJECTION;;@ /@/@|?@
INJECTION;;@ @|@
INJECTION;;@|@
INJECTION;? -n @:\PATH;//@
INJECTION;?%0a@
INJECTION;?@_@=@&@=|
INJECTION;?@=%0a@&@=@
INJECTION;?@=;&@=/@/
INJECTION;?@=@&@=;@
INJECTION;?|@
INJECTION;@ /@/@@@@@@
INJECTION;@ /@/@|?@=
INJECTION;@ @:\@.ini
INJECTION;@&@_@= ;@;
INJECTION;@&@=;@ /@/
INJECTION;@&@=;@ /@|
INJECTION;@&@=@&@=|/
INJECTION;@&@=|/@/@|
INJECTION;@&@=|@ /@/
INJECTION;@&@=+'@"=
INJECTION;@'(@ @
INJECTION;@.@.org |
INJECTION;@.@/@=;@ @
INJECTION;@.bat?&@
INJECTION;@.bat?&@
INJECTION;@.bat?&@+@
INJECTION;@.bat?|@ .
INJECTION;@.bat|@ @:
INJECTION;@.cgi?'%0a/@
INJECTION;@.cgi?@=%0a@
INJECTION;@.cgi?@=;&
INJECTION;@.EXE? -@
INJECTION;@.exe|@
INJECTION;@.exe|@ @:
INJECTION;@/@ /@/@|
INJECTION;@/@&@_@= ;
INJECTION;@/@.@/@=;@
INJECTION;@/@.bat|@
INJECTION;@/@;@ /@/@
INJECTION;@/@|&@_@=@
INJECTION;@/@=;@ @.@
INJECTION;@;@|@
INJECTION;@?/c+@
INJECTION;@?@%0a@
INJECTION;@?@=&@=;@
INJECTION;@?@=;@ /@
INJECTION;@?@=@&@=;@
INJECTION;@?|@
INJECTION;@?|=@
INJECTION;@@ @ /@/@&
INJECTION;-@-@&@=";@
INJECTION;@-@&@=";@
INJECTION;@-@&@=;@ /
INJECTION;@@@ @ /@/@
INJECTION;@-@-@&@=";
INJECTION;@_@=@';@@@
INJECTION;@|&@_@=@&@
INJECTION;@|@
INJECTION;@|@ " @" @
INJECTION;@+@+@:\+/@
INJECTION;@= @ /@/@@
INJECTION;@=" -@|@ "
INJECTION;@=";@ ";
INJECTION;@=";@ ";
INJECTION;@=%0a@
INJECTION;@=/@/@&@=;
INJECTION;@=;&@
INJECTION;@=;@ /@/@@
INJECTION;@=;@ /@/@|
INJECTION;@=;@"@
INJECTION;@=;@|"@
INJECTION;@=;@|@
INJECTION;@=@ @ @ @_
INJECTION;@=@ @@@@@@

INJECTION;@=@";@
INJECTION;@=@%0a@
INJECTION;@=@&@=;@ /
INJECTION;@=@&@=|@ /
INJECTION;@=@&@=|@ @
INJECTION;@=@&@=+'@'
INJECTION;@=@;@|@
INJECTION;@=@-@&@=;@
INJECTION;@=@@@ @ /@
INJECTION;@='@@@@@@@@
INJECTION;@=@_@&@=;@
INJECTION;@=@|&@
INJECTION;@=@|@
INJECTION;@=@+@:@.@.
INJECTION;@=`/@/@`@@
INJECTION;@=`@ @` @
INJECTION;@=|/@/@/@|
INJECTION;@=|@
INJECTION;@=|@ /@/@|
INJECTION;@=|@|@@@@@@
INJECTION;@"='+&@=@
INJECTION;@=+'@"='+
INJECTION;_@=@&@_@=<
INJECTION;|&@_@=@&@=
INJECTION;|@ " @" @
INJECTION;|@ /@/@|&@
INJECTION;|@ @:\@\@.
INJECTION;=" -@|@ "
INJECTION;=";@
INJECTION;=";@ "; @
INJECTION;=%0a@
INJECTION;=&@=;@ /@/
INJECTION;=;&@
INJECTION;=;&@=/@/@
INJECTION;=;@ /@/@|&
INJECTION;=;@ @|@
INJECTION;=;@"@
INJECTION;=;@|"@
INJECTION;=@;@|@
INJECTION;=@|&@
INJECTION;=`@` @
INJECTION;=|@
INJECTION;at|@ @:\@\
INJECTION;bat|@ @:\@
INJECTION;cgi?@=%0a/
INJECTION;cgi?@=&@=;
INJECTION;t|@ @:\@\@
NORMAL;&@_@=@&@_@
NORMAL;&@_@=@.txt
NORMAL;&@=/@/&@=@
NORMAL;&@=/@/@&@_
NORMAL;&@=@&@_@=@
NORMAL;&@=@&@=/@/
NORMAL;&@=@_@&@=/
NORMAL;&@=@_@&@=@
NORMAL;./@&@=@&@=
NORMAL;./@/@&@=@&
NORMAL;./@/@.ini
NORMAL;./@/@/@.ex
NORMAL;.@/@/@.cgi
NORMAL;.@/@/@/@.
NORMAL;.\@\@.ini&
NORMAL;.cgi?@_@=@
NORMAL;.cgi?@=&@=
NORMAL;.exe/@_@.e
NORMAL;.php&@_@=@
NORMAL;.php?@=@/@
NORMAL;.txt&@=/@/
NORMAL;/&@=@.php&
NORMAL;/@&@=@&@=@
NORMAL;/@&@=@@@&@
NORMAL;/@.@ @/
NORMAL;/@.@.@.@.@
NORMAL;/@.@.@.php
NORMAL;/@.@.@.php
NORMAL;/@.@/@@@@@
NORMAL;/@.@@@@@@@@
NORMAL;/@.asa@@@@
NORMAL;/@.asp.@@@
NORMAL;/@.asp@@@@
NORMAL;/@.bat@@@@
NORMAL;/@.cfm?@=@
NORMAL;/@.cfm@@@@
NORMAL;/@.cgi?@=@
NORMAL;/@.cgi@@@@
NORMAL;/@.dbm@@@@
NORMAL;/@.dll/_@_
NORMAL;/@.exe/@@@
NORMAL;/@.exe/@_@
NORMAL;/@.exe/_@_
NORMAL;/@.exe?@=@
NORMAL;/@/&@=@.ph
NORMAL;/@/@&@_@=
NORMAL;/@/@&@=@&@
NORMAL;/@/@&@=@@
NORMAL;/@/@&@=@@@
NORMAL;/@/@.@.@.@
NORMAL;/@/@.@.@.p
NORMAL;/@/@.@@@@@
NORMAL;/@/@.cfm?@
NORMAL;/@/@.ini@@
NORMAL;/@/@.mdb@@
NORMAL;/@/@.nsf@@
NORMAL;/@/@.php@@
NORMAL;/@/@.php@@
NORMAL;/@/@.txt@@
NORMAL;/@/@/@.@.@
NORMAL;/@/@/@.@.@
NORMAL;/@/@/@.@/@
NORMAL;/@/@/@.@@@
NORMAL;/@/@-@@@@@
NORMAL;/@/@@@@@@@@
NORMAL;/@/@_@.@@@
NORMAL;/@/@_@.@_@
NORMAL;/@/@_@.asp
NORMAL;/@/@_@.cgi
NORMAL;/@/@_@.htm
NORMAL;/@/@_@.log
NORMAL;/@/@_@.nsf
NORMAL;/@/@_@.php
NORMAL;/@/@_@/@.p
NORMAL;/@/@_@@@@@
NORMAL;/@/@_@_@.p
NORMAL;/@/@_@_@_@
NORMAL;/@/@=@.@&@
NORMAL;/@-@.@-@.h
NORMAL;/@-@/.@/@/
NORMAL;/@-@/@.@/@
NORMAL;/@-@/@.@@@
NORMAL;/@-@/@.asp
NORMAL;/@-@/@.bat
NORMAL;/@-@/@.cfm
NORMAL;/@-@/@.cgi
NORMAL;/@-@/@.exe
NORMAL;/@-@/@.inc
NORMAL;/@-@/@.php
NORMAL;/@-@/@.plx
NORMAL;/@-@/@.txt
NORMAL;/@-@/@_@.@
NORMAL;/@-@/@_@.c
NORMAL;/@-@/@_@.p
NORMAL;/@-@/@_@/@
NORMAL;/@-@/@_@@@
NORMAL;/@-@/@_@_@
NORMAL;/@-@-@.@@@
NORMAL;/@-@-@.jsp
NORMAL;/@-@@@@@@@@

NORMAL;/@@@@@@@@
NORMAL;/@-@@@@@@@
NORMAL;/@@@@@@@@@
NORMAL;@&@[@][@]=
NORMAL;@&@_@=@&@_
NORMAL;@&@=/@/@&@
NORMAL;@&@=@&@_@=
NORMAL;@&@=@&@=/@
NORMAL;@&@=@&@=@-
NORMAL;@&@=@&@=@&
NORMAL;@&@=@&@=@_
NORMAL;@&@=@.@&@_
NORMAL;@&@=@.php&
NORMAL;@.@.@/@.js
NORMAL;@.@?@=@&@=
NORMAL;@.@?@=@/@/
NORMAL;@.cgi?@_@=
NORMAL;@.cgi?@=&@
NORMAL;@.cgi?@=@&
NORMAL;@.dll/_@_@
NORMAL;@.exe/@_@.
NORMAL;@.exe/_@_@
NORMAL;@.exe?@=@&
NORMAL;@.php&@_@=
NORMAL;@.php?@=@/
NORMAL;@.txt&@=/@
NORMAL;@/&@=@.php
NORMAL;-@/.@/@/@.
NORMAL;-@/@/@.php
NORMAL;-@/@/@.txt
NORMAL;-@/@/@.xml
NORMAL;-@/@/@/@.@
NORMAL;@/@/@/@.@/
NORMAL;@/@/@/@.as
NORMAL;@/@/@/@.e
NORMAL;@/@/@/@.ex
NORMAL;-@/@/@/@/@
NORMAL;@/@?@=@&@=
NORMAL;@/@@&@_@=
NORMAL;-@/@-@.@/@
NORMAL;-@/@-@.cgi
NORMAL;@/@-@.inc.
NORMAL;-@/@-@/@.p
NORMAL;@/@-@/@.ph
NORMAL;@/@-@/@/@.
NORMAL;@/@-@/@-@.
NORMAL;-@/@@@@@@@@
NORMAL;@/@_@.@_@_
NORMAL;-@/@_@.cgi
NORMAL;-@/@_@.php
NORMAL;-@/@_@/@.p
NORMAL;@/@_@/@.ph
NORMAL;@/@_@_@.ph
NORMAL;-@/@_@_@_@
NORMAL;@/@_@_@_@.
NORMAL;@/@=@.@&@=
NORMAL;@?@=@&@=@&
NORMAL;@?@=@&@=@/
NORMAL;@-@&@=@@@@
NORMAL;@-@.@-@.ht
NORMAL;@-@.cgi?@=
NORMAL;@-@/.@/@/@
NORMAL;@-@/@.@/@=
NORMAL;@-@/@.cgi/
NORMAL;-@@@@@@@@@@
NORMAL;@@@@@@@@@@@
NORMAL;@-@=@@@@@@
NORMAL;@[@][@]=@&
NORMAL;@\@.ini&@=
NORMAL;@][@]=@&@=
NORMAL;@]=@&@=@&@
NORMAL;@_@&@=/@/@
NORMAL;@_@&@=@&@=
NORMAL;@_@&@=@_@&
NORMAL;@_@.@?@=@
NORMAL;@_@.cgi?@=
NORMAL;@_@.exe?@=
NORMAL;@_@/@.dll/
NORMAL;@_@/@.exe/
NORMAL;@_@/@/@.cg
NORMAL;@_@/@/@.ph
NORMAL;@_@/@/@_@_
NORMAL;@_@/_@_@/@
NORMAL;@_@_@.@.cg
NORMAL;@_@_@_@_@.@.
NORMAL;@_@=/@/@@
NORMAL;@_@=@&@_@=
NORMAL;@_@=@&@=@&
NORMAL;@_@=@&@=@.
NORMAL;@_@=@.txt&
NORMAL;@_@=@://@/
NORMAL;@_@=@://@@
NORMAL;@_@=@@@@@@@
NORMAL;@=*&@=@&@=
NORMAL;@=*&@=@@@@
NORMAL;@=/@.asp@@
NORMAL;@=/@/&@=@.
NORMAL;@=/@/@&@_@
NORMAL;@=/@/@@@@
NORMAL;@=/@/@@@@@
NORMAL;@=;@=@&@=@
NORMAL;@=@&@[@][@
NORMAL;@=@&@_@=@&
NORMAL;@=@&@=/@/@
NORMAL;@=@&@=@&@=
NORMAL;@=@&@=@.@&
NORMAL;@=@&@=@.ph
NORMAL;@=@&@=@://
NORMAL;@=@&@=@-@-
NORMAL;@=@&@=@@@@
NORMAL;@=@&@=@_@&
NORMAL;@=@&@=@_@&
NORMAL;@=@.@&@_@_
NORMAL;@=@.@&@=/@
NORMAL;@=@.org
NORMAL;@=@.php&@_
NORMAL;@=@.txt&@=
NORMAL;@=@://@/@@
NORMAL;@=@://@@@@
NORMAL;@=@-@-@&@=
NORMAL;@=@-@@@@@@
NORMAL;@=@@@@@@@@@
NORMAL;@=@@@@@@@@@
NORMAL;@=@_@&@=/.
NORMAL;@=@_@&@=/@
NORMAL;@=@_@&@=@&
NORMAL;@=@_@&@=@_
NORMAL;@=@_@@@@@@
NORMAL;[@][@]=@&@
NORMAL;[@]=@&@=@&
NORMAL;\@.ini&@=.
NORMAL;\@.ini&@=@
NORMAL;\@\@.ini&@
NORMAL;_@&@=/@/@&
NORMAL;_@&@=@&@_@
NORMAL;_@&@=@&@=@
NORMAL;_@&@=@_@&@
NORMAL;_@/@.dll/_
NORMAL;_@/@.exe/@
NORMAL;_@/@.exe/_
NORMAL;_@/@/@.cgi
NORMAL;_@/@/@.php
NORMAL;_@/@/@_@_@
NORMAL;_@/_@_@/@.
NORMAL;_@_@.@.cgi
NORMAL;_@_@/@.dll
NORMAL;_@_@/@.exe

NORMAL;_@_@/@.pwd
NORMAL;_@_@/_@_@/
NORMAL;_@_@\@.cnf
NORMAL;_@_@_@.@.c
NORMAL;_@=@&@=@&@
NORMAL;_@=@&@=@.@
NORMAL;_@=@.txt&@
NORMAL;=*&@=@&@=.
NORMAL;=/@/&@=@.p
NORMAL;=/@/@&@_@=
NORMAL;=@&@_@=@&@
NORMAL;=@&@=@.@&@
NORMAL;=@&@=@.php
NORMAL;=@&@=@://@
NORMAL;=@&@=@-@-@
NORMAL;=@&@=@_@&@
NORMAL;=@.@&@_@_@
NORMAL;=@.@&@=/@/
NORMAL;=@.php&@_@
NORMAL;=@.txt&@=/
NORMAL;=@-@-@&@="
NORMAL;=@_@&@=/@/
NORMAL;=@_@&@=@&@
NORMAL;=@_@&@=@_@
NORMAL;asp &@=@&@
NORMAL;cgi?@=@&@=
NORMAL;exe?@=@&@=
NORMAL;hp&@_@=@.t
NORMAL;jsp?@=@&@=
NORMAL;p &@=@&@=@
NORMAL;p&@_@=@.tx
NORMAL;php&@_@=@.
NORMAL;php?@=@&@=
NORMAL;php?@=@/@.
NORMAL;sp &@=@&@=
NORMAL;t&@=/@/&@=
NORMAL;txt&@=/@/&
NORMAL;xt&@=/@/&@
PATH;"./."./@/@
PATH;"./@/@.ini
PATH;%@..%@..%@
PATH;%@..%@..%@
PATH;%@..%@../@
PATH;%@..%@../@
PATH;%@..%@/@/@
PATH;%@../@/@/@
PATH;%@../@/@/@
PATH;%@/@/@/@.e
PATH;%00/@
PATH;%00html
PATH;&@=../../.
PATH;&@=..\..\.
PATH;&@=/../../
PATH;&@=@&@=../
PATH;*&@=@&@=..
PATH;*.@//@.jsp
PATH;*//../../.
PATH;."./."./@/
PATH;."./@/@.in
PATH;.%@..%@..%
PATH;.%@..%@..%
PATH;.%@..%@../
PATH;.%00@
PATH;..%@..%@..
PATH;...box//..
PATH;...nsf/../
PATH;...nsf//..
PATH;../../../.
PATH;../../..//
PATH;../../../@
PATH;../../@/@/
PATH;../../@/@/
PATH;..//.../.
PATH;..//.../@
PATH;../@
PATH;../@%00@
PATH;../@&@=@&@
PATH;../@/@ /@/
PATH;../@/@&@=.
PATH;../@/@&@=@
PATH;..@/../../
PATH;..@//../..
PATH;..@/@/@/@
PATH;..\..\..\.
PATH;..\..\..\.
PATH;..\..\..\@
PATH;..\..\@
PATH;..\..\@.in
PATH;..\..\@\@.
PATH;..\@
PATH;..\@\@.ini
PATH;..\\@
PATH;..€Ç¯../@/
PATH;..€Ç¯..€Ç¯
PATH;..Â../@/
PATH;..box//../
PATH;..nsf/../.
PATH;..nsf//../
PATH;./."./@/@.
PATH;./../../..
PATH;./../../..
PATH;./../../..//@
PATH;./../../@&
PATH;./../../@.
PATH;./../../@/
PATH;./../../@/
PATH;./../@&@=@
PATH;./.|./.|./
PATH;.//...//..
PATH;.//...//@.
PATH;.//@
PATH;./@/@ /@/@
PATH;./@/@&@=..
PATH;.@../@/@/
PATH;.@..@..
PATH;.@..@../
PATH;.@..@/@/
PATH;.@/../../.
PATH;.@//../../
PATH;.@=../../.
PATH;.\..\..\..
PATH;.\..\..\..\@.
PATH;.Â□..Â□../
PATH;.Â□..Â□../
PATH;.Á..Á..Á
PATH;/..%@..%@.
PATH;/..%@..%@.
PATH;/...//.../
PATH;/...//@.ba
PATH;/...//@.in
PATH;/...@/../.
PATH;/...@//../
PATH;/...box//.
PATH;/...nsf/..
PATH;/...nsf//.
PATH;/../../../
PATH;/../@/@/@.
PATH;/../@/@/@_
PATH;/../<@>@('
PATH;/..@..@.
PATH;/..\\@
PATH;/..€Ç¯..€Ç
PATH;/..Á..Á.
PATH;/..Â..Â.
PATH;/..Â□..Â□.
PATH;/..Â□..Â□.
PATH;/..Á/../@/

PATH;/..Á/..Á/.
PATH;//.%@/.%@/
PATH;//...//...
PATH;//...//@.b
PATH;//...//@.i
PATH;//../../..
PATH;//////@
PATH;/@/../../.
PATH;/@/../@/@/
PATH;/@/..@..
PATH;/@/\..\..\
PATH;/@-@/..%@.
PATH;/@-@/.../.
PATH;/@-@/..@.
PATH;/@-@//.%@/
PATH;/@-@//@/@/
PATH;@..%@..%@/
PATH;@..%@..%@/
PATH;@..%@../@.
PATH;@..%@../@/
PATH;@..%@/@/@/
PATH;@..%2f@
PATH;@..%5c@
PATH;@.../@
PATH;@..@../@
PATH;@..@..@/
PATH;@..@/@/@
PATH;@..@/@/@/
PATH;@..\..\@
PATH;@..\@
PATH;@..\\@
PATH;@.//@
PATH;@.@=../../
PATH;@.|./@
PATH;@.ini&@=..
PATH;@.ntf+++++
PATH;@.php @
PATH;@/%00@
PATH;@/*//../..
PATH;@/.%@/.%@/
PATH;@/.%@/./@/
PATH;@/.%@/@/@.
PATH;@/.%@/@/@/
PATH;-@/..%@..%
PATH;@/..%@..%@
PATH;-@/.../..
PATH;@/.../...
PATH;@/../../..
PATH;@/../@/@/@
PATH;-@/..@..
PATH;@/..@..@
PATH;@/..\\@
PATH;@/..€Ç¯..€
PATH;@/..Â£..Â£
PATH;@/..ÂÂ..ÂÂ
PATH;@/./@/@/@.
PATH;-@//.%@/.%
PATH;@//.%@/.%@
PATH;@//../../.
PATH;@//////@
PATH;@///@
PATH;@//@
PATH;-@//@/@/@.
PATH;@-@/.../.
PATH;@-@/..@..
PATH;@-@//.%@/.
PATH;@-@-@=..\.
PATH;-@-@=../..
PATH;@-@=../../
PATH;-@-@=..\..
PATH;@-@=..\..\
PATH;@[@]=../..
PATH;@]=../../.
PATH;@_/@/..%@.
PATH;@_/@/..%@.
PATH;-@=..\..\.
PATH;@=..\..\..
PATH;@=.|./.|./
PATH;@=/&@=../.
PATH;@=/../../.
PATH;@=/../../.
PATH;@=/../@
PATH;@=@&@=../.
PATH;@=@&@=..\.
PATH;@=@:\&@=@&
PATH;[@]=../../
PATH;\..\..\..\
PATH;\..\..\@.i
PATH;\..\..\@\@
PATH;\..\@\@.in
PATH;]/..%@..%@
PATH;_/@/..%@.@
PATH;_/@/..%@.@
PATH;_@&@=/../.
PATH;_@/..%@..%
PATH;_@_@=.|./.
PATH;_@=../../.
PATH;_@=.|./.|.
PATH;_@=@&@=../
PATH;_@=@&@=..\
PATH;|.%2F.|@
PATH;|./.|./.|.
PATH;|./.|./@/@
PATH;|@..\\@
PATH;¯../@/@/@.
PATH;¯..€Ç¯../@
PATH;¯..€Ç¯..€Ç
PATH;¯..Á¯../@/
PATH;¯..Á¯..Á¯.
PATH;++++++.nsf
PATH;++++++++++
PATH;=@&@=..\..
PATH;=@:\&@=@&@
PATH;=@:\@\@.in
PATH;=@_@&@=/..
PATH;£../@/@/@.
PATH;Â..ÂÂ..ÂÂ.
PATH;Á/../@/@/@
PATH;Á/..Á/../@
PATH;Á/..Á/..Á/
PATH;Á¯../@/@/@
PATH;Á¯..Á¯../@
PATH;Á¯..Á¯..Á¯
PATH;Â£../@/@/@
PATH;Â£..Â£../@
PATH;Â£..Â£..Â£
PATH;ÂÂ../@/@/@
PATH;ÂÂ..ÂÂ../@
PATH;ÂÂ..ÂÂ..ÂÂ
PATH;box//../..
PATH;Ç¯../@/@/@
PATH;Ç¯..€Ç¯../
PATH;Ç¯..€Ç¯..€
PATH;exe/@_@.ex
PATH;f/../../..
PATH;f//../../.
PATH;f+++++++++
PATH;i&@=..\..\
PATH;ini&@=..\.
PATH;ini.@.txt
PATH;ni&@=..\..
PATH;nsf/../../
PATH;nsf//../..
PATH;ntf+++++++
PATH;ox//../../
PATH;php&@=../.
PATH;php?@=@://@

PATH;sf/../../.
PATH;sf//../../
PATH;x//../../.
PATH;xe/@_@.exe
SQL;' --@
SQL;' @ @
SQL;' @ 1==1--@
SQL;' group @
SQL;' having @
SQL;' OR 1==1--@
SQL;' UNION @
SQL;'; @ @
SQL;'; insert @
SQL;'; SELECT @
SQL;'union @ select
XSS;")</@>.jsp
XSS;")</@>/@.@
XSS;"@")</@>/@
XSS;"@:"> <@:@
XSS;"@:"><@:@>
XSS;"@\")</@>.
XSS;"><@>@(@)<
XSS;&@;@&@;@('
XSS;&@=<@>@</@
XSS;&@=<@>@=@;
XSS;("@")</@>/
XSS;()</@:@></
XSS;('@');&@;/
XSS;('@');</@>
XSS;('@')</@>.
XSS;(@)</@><!-
XSS;.@)</@>&@=
XSS;.jsp/<@>@(
XSS;/&@;@&@;@(
XSS;/@.jsp/<@>
XSS;/@/@/<@>@(
XSS;/@/<@>@(@.
XSS;/@:@></@:@
XSS;/</@><@>@(
XSS;/<@>@("@")
XSS;/<@>@('@')
XSS;:<@>@('@')
XSS;;@&@;@('@'
XSS;;@('@');&@
XSS;;</@>&@=@&
XSS;;</@>&@=<@
XSS;@ @()</@:
XSS;@ @()</@:@
XSS;@ @="@.@"?
XSS;@" @ @()</
XSS;@")</@>/@.
XSS;@&@;@('@')
XSS;@&@_@=<@>@
XSS;@&@="><@>@
XSS;@&@=@&@=<@
XSS;@&@=@:<@>@
XSS;@&@=<@>@.@
XSS;@&@=<@>@</
XSS;@&@=<@>@=@
XSS;@(@.@)</@>
XSS;@(\"@\")</
XSS;@.@);</@>&
XSS;@.@)</@>&@
XSS;'@;@
XSS;`@ `;@();@
XSS;|@||/@|
XSS;~/<@>@(@.@
XSS;</@:@></@
XSS;</@:@></@:
XSS;</@>&@=@&@
XSS;</@>&@=<@>
XSS;<@:@> <@
XSS;<@:@> <@:
XSS;<@:@>@ "@:
XSS;<@>@('@')<
XSS;=@&@="><@>
XSS;=@&@=@:<@>
XSS;=@&@=<@>@.
XSS;=@&@=<@>@<
XSS;=@&@=<@>@=
XSS;=@:<@>@('@
XSS;=@;</@>&@=
XSS;=<@>@(@)</
XSS;=<@>@(@.@)
XSS;=<@>@.@(@.
XSS;=<@>@.@</@
XSS;=<@>@=@;</
XSS;p/<@>@('@'
XSS;sp/<@>@('@

ANEXO 4: STRINGS.CONF

El archivo Strings.conf contiene las cadenas que deben ser revisadas en todos los requerimientos verificando su presencia. Si existe aluna cadena en el requerimiento es señal de un intento por obtener un recurso que generalmente no debe estar disponible al publico y por lo tanto corresponde a un intento de ataque. El siguiente es un ejemplo básico de dicho archivo:

```
#####################################################################
#IDS Strings.conf
#####################################################################
#Cadenas que generalmente no deben aparecer en un requerimiento.
#####################################################################
/etc/passwd
/etc/shadow
/bin/sh
/bin/bash
/bin/tcsh
global.asa
/~root
boot.ini
win.ini
sam._
#EoF
```

ANEXO 5: EJEMPLO IDS.CONF

Archivo de configuración principal del prototipo.

```
#####################################################
#IDS.CONF
#Generic configuration.(generado por idsconfig)
#####################################################
#Files
#####################################################
LOG_FILE= /usr/local/squid/ids/log/ids.log
LIMITS_FILE= /usr/local/squid/ids/log/limits.log
STRINGS_FILE= /usr/local/squid/ids/etc/strings.conf
#####################################################
#Levels
#####################################################
#Check the presense of suspicious string
CHECK_STRINGS= 1
#Check size limits of requests
CHECK_LIMITS= 1
#Checking using NN
CHECK_NN= 1
#Generates limits
GEN_LIMITS= 1
#Log entire request if attack found
DETAIL= 1
#Generate Limits even if an attack is found
#(WARNING)
OVERRIDE_GENLIM= 0
#####################################################
#Limits
#Parameter: min,max
#####################################################
GENERIC_REQUEST_SIZE: 560,731
METHOD_SIZE: 3,4
URI_GET_SIZE: 32,200
URI_POST_SIZE: 40,40
HEADERS_SIZE: 484,656
HEADER_Connection_SIZE: 10,200
HEADER_User-Agent_SIZE: 62,62
HEADER_Host_SIZE: 14,23
BODY_SIZE: 0,23
PARAMETERS: 0,8
```

ANEXO 6: EJEMPLO DE LIMITS.LOG

Este es un pequeño ejemplo de los datos almacenados en el archivo limits.log, el cual contiene las características en tamaño de cada parte del requerimiento analizado. Estos datos son utilizados para determinar los límites normales y específicos de los requerimientos que deben ser usados en el prototipo.

+Fri Apr 25 11:01:20 2003+ http://fing.javeriana.edu.co/
[Fri Apr 25 11:01:20 2003] GENERIC_REQUEST_SIZE: 545
[Fri Apr 25 11:01:20 2003] METHOD_SIZE: 3
[Fri Apr 25 11:01:20 2003] URI_GET_SIZE: 29
[Fri Apr 25 11:01:20 2003] VERSION_SIZE: 8
[Fri Apr 25 11:01:20 2003] HEADERS_SIZE: 497
[Fri Apr 25 11:01:20 2003] HEADER_Host_SIZE: 21
[Fri Apr 25 11:01:20 2003] HEADER_User-Agent_SIZE: 62
[Fri Apr 25 11:01:20 2003] HEADER_Accept_SIZE: 147
[Fri Apr 25 11:01:20 2003] HEADER_Accept-Language_SIZE: 16
[Fri Apr 25 11:01:20 2003] HEADER_Accept-Encoding_SIZE: 29
[Fri Apr 25 11:01:20 2003] HEADER_Accept-Charset_SIZE: 34
[Fri Apr 25 11:01:20 2003] HEADER_Keep-Alive_SIZE: 3
[Fri Apr 25 11:01:20 2003] HEADER_Proxy-Connection_SIZE: 10
[Fri Apr 25 11:01:20 2003] HEADER_Cookie_SIZE: 45
[Fri Apr 25 11:01:20 2003] BODY_SIZE: 0
+Fri Apr 25 11:01:21 2003+ http://fing.javeriana.edu.co/ingenieria/noticias
[Fri Apr 25 11:01:21 2003] GENERIC_REQUEST_SIZE: 564
[Fri Apr 25 11:01:21 2003] METHOD_SIZE: 3
[Fri Apr 25 11:01:21 2003] URI_GET_SIZE: 48
[Fri Apr 25 11:01:21 2003] VERSION_SIZE: 8
[Fri Apr 25 11:01:21 2003] HEADERS_SIZE: 497
[Fri Apr 25 11:01:21 2003] HEADER_Host_SIZE: 21
[Fri Apr 25 11:01:21 2003] HEADER_User-Agent_SIZE: 62
[Fri Apr 25 11:01:21 2003] HEADER_Accept_SIZE: 147
[Fri Apr 25 11:01:21 2003] HEADER_Accept-Language_SIZE: 16
[Fri Apr 25 11:01:21 2003] HEADER_Accept-Encoding_SIZE: 29
[Fri Apr 25 11:01:21 2003] HEADER_Accept-Charset_SIZE: 34
[Fri Apr 25 11:01:21 2003] HEADER_Keep-Alive_SIZE: 3
[Fri Apr 25 11:01:21 2003] HEADER_Proxy-Connection_SIZE: 10
[Fri Apr 25 11:01:21 2003] HEADER_Cookie_SIZE: 45
[Fri Apr 25 11:01:21 2003] BODY_SIZE: 0
+Fri Apr 25 11:01:21 2003+ http://fing.javeriana.edu.co/ingenieria/noticias/
[Fri Apr 25 11:01:21 2003] GENERIC_REQUEST_SIZE: 565
[Fri Apr 25 11:01:21 2003] METHOD_SIZE: 3

[Fri Apr 25 11:01:21 2003] URI_GET_SIZE: 49
[Fri Apr 25 11:01:21 2003] VERSION_SIZE: 8
[Fri Apr 25 11:01:21 2003] HEADERS_SIZE: 497
[Fri Apr 25 11:01:21 2003] HEADER_Host_SIZE: 21
[Fri Apr 25 11:01:21 2003] HEADER_User-Agent_SIZE: 62
[Fri Apr 25 11:01:21 2003] HEADER_Accept_SIZE: 147
[Fri Apr 25 11:01:21 2003] HEADER_Accept-Language_SIZE: 16
[Fri Apr 25 11:01:21 2003] HEADER_Accept-Encoding_SIZE: 29
[Fri Apr 25 11:01:21 2003] HEADER_Accept-Charset_SIZE: 34
[Fri Apr 25 11:01:21 2003] HEADER_Keep-Alive_SIZE: 3
[Fri Apr 25 11:01:21 2003] HEADER_Proxy-Connection_SIZE: 10
[Fri Apr 25 11:01:21 2003] HEADER_Cookie_SIZE: 45
[Fri Apr 25 11:01:21 2003] BODY_SIZE: 0
+Fri Apr 25 11:01:23 2003+
http://fing.javeriana.edu.co/ingenieria/banners/random7.jpg
[Fri Apr 25 11:01:23 2003] GENERIC_REQUEST_SIZE: 635
[Fri Apr 25 11:01:23 2003] METHOD_SIZE: 3
[Fri Apr 25 11:01:23 2003] URI_GET_SIZE: 59
[Fri Apr 25 11:01:23 2003] VERSION_SIZE: 8
[Fri Apr 25 11:01:23 2003] HEADERS_SIZE: 557
[Fri Apr 25 11:01:23 2003] HEADER_Host_SIZE: 21
[Fri Apr 25 11:01:23 2003] HEADER_User-Agent_SIZE: 62
[Fri Apr 25 11:01:23 2003] HEADER_Accept_SIZE: 147
[Fri Apr 25 11:01:23 2003] HEADER_Accept-Language_SIZE: 16
[Fri Apr 25 11:01:23 2003] HEADER_Accept-Encoding_SIZE: 29
[Fri Apr 25 11:01:23 2003] HEADER_Accept-Charset_SIZE: 34
[Fri Apr 25 11:01:23 2003] HEADER_Keep-Alive_SIZE: 3
[Fri Apr 25 11:01:23 2003] HEADER_Proxy-Connection_SIZE: 10
[Fri Apr 25 11:01:23 2003] HEADER_Referer_SIZE: 49
[Fri Apr 25 11:01:23 2003] HEADER_Cookie_SIZE: 45
[Fri Apr 25 11:01:23 2003] BODY_SIZE: 0
+Fri Apr 25 11:01:26 2003+
http://fing.javeriana.edu.co/ingenieria/noticias/galeria/borde_titulo_2.jpg
[Fri Apr 25 11:01:26 2003] GENERIC_REQUEST_SIZE: 739
[Fri Apr 25 11:01:26 2003] METHOD_SIZE: 3
[Fri Apr 25 11:01:26 2003] URI_GET_SIZE: 75
[Fri Apr 25 11:01:26 2003] VERSION_SIZE: 8
[Fri Apr 25 11:01:26 2003] HEADERS_SIZE: 645
[Fri Apr 25 11:01:26 2003] HEADER_Host_SIZE: 21
[Fri Apr 25 11:01:26 2003] HEADER_User-Agent_SIZE: 62
[Fri Apr 25 11:01:26 2003] HEADER_Accept_SIZE: 147
[Fri Apr 25 11:01:26 2003] HEADER_Accept-Language_SIZE: 16
[Fri Apr 25 11:01:26 2003] HEADER_Accept-Encoding_SIZE: 29
[Fri Apr 25 11:01:26 2003] HEADER_Accept-Charset_SIZE: 34
[Fri Apr 25 11:01:26 2003] HEADER_Keep-Alive_SIZE: 3
[Fri Apr 25 11:01:26 2003] HEADER_Proxy-Connection_SIZE: 10

[Fri Apr 25 11:01:26 2003] HEADER_Referer_SIZE: 49
[Fri Apr 25 11:01:26 2003] HEADER_Cookie_SIZE: 45
[Fri Apr 25 11:01:26 2003] HEADER_If-Modified-Since_SIZE: 29
[Fri Apr 25 11:01:26 2003] HEADER_If-None-Match_SIZE: 21
[Fri Apr 25 11:01:26 2003] BODY_SIZE: 0
+Fri Apr 25 11:01:26 2003+
http://fing.javeriana.edu.co/ingenieria/noticias/galeria/borde_titulo.jpg
[Fri Apr 25 11:01:26 2003] GENERIC_REQUEST_SIZE: 737
[Fri Apr 25 11:01:26 2003] METHOD_SIZE: 3
[Fri Apr 25 11:01:26 2003] URI_GET_SIZE: 73
[Fri Apr 25 11:01:26 2003] VERSION_SIZE: 8
[Fri Apr 25 11:01:26 2003] HEADERS_SIZE: 645
[Fri Apr 25 11:01:26 2003] HEADER_Host_SIZE: 21
[Fri Apr 25 11:01:26 2003] HEADER_User-Agent_SIZE: 62
[Fri Apr 25 11:01:26 2003] HEADER_Accept_SIZE: 147
[Fri Apr 25 11:01:26 2003] HEADER_Accept-Language_SIZE: 16
[Fri Apr 25 11:01:26 2003] HEADER_Accept-Encoding_SIZE: 29
[Fri Apr 25 11:01:26 2003] HEADER_Accept-Charset_SIZE: 34
[Fri Apr 25 11:01:26 2003] HEADER_Keep-Alive_SIZE: 3
[Fri Apr 25 11:01:26 2003] HEADER_Proxy-Connection_SIZE: 10
[Fri Apr 25 11:01:26 2003] HEADER_Referer_SIZE: 49
[Fri Apr 25 11:01:26 2003] HEADER_Cookie_SIZE: 45
[Fri Apr 25 11:01:26 2003] HEADER_If-Modified-Since_SIZE: 29
[Fri Apr 25 11:01:26 2003] HEADER_If-None-Match_SIZE: 21
[Fri Apr 25 11:01:26 2003] BODY_SIZE: 0
…

ANEXO 7. EJEMPLO ARCHIVO ENTRENAMIENTO RED NEURONAL

NOTA: Este archivo es generado automáticamente en base a los patrones básicos por medio del programa data7train.c el cual recibe un archivo de patrones básicos y los convierte al formato manejado por SNNS.

SNNS pattern definition file V3.2
generated at Tue Aug 3 00:00:44 2003
No. of patterns : 378
No. of input units : 80
No. of output units : 5

#Input 1
0 0 1 0 0 1 1 0
1 1 1 1 1 1 1 1
0 0 1 1 1 1 0 1
0 0 1 0 0 0 1 0
0 0 1 1 1 0 1 1
1 1 1 1 1 1 1 1
0 0 1 0 0 0 0 0
0 0 1 0 0 1 1 1
0 0 1 0 0 1 1 1
0 0 1 1 1 0 1 1
#Output 1 0
1 0 0 0 0
#Input 2
0 0 1 0 0 1 1 0
1 1 1 1 1 1 1 1
0 0 1 1 1 1 0 1
0 0 1 1 1 0 1 1
1 1 1 1 1 1 1 1
0 0 1 0 0 0 0 0
0 0 1 0 1 1 1 1
1 1 1 1 1 1 1 1
0 0 1 0 1 1 1 1
1 1 1 1 1 1 1 1
#Output 2 0
1 0 0 0 0
#Input 3
0 0 1 0 0 1 1 1

0 0 1 0 0 1 1 1
0 0 1 1 1 0 1 1
0 0 1 0 0 0 0 0
1 1 1 1 1 1 1 1
0 0 1 0 0 0 0 0
0 1 1 0 0 0 0 0
1 1 1 1 1 1 1 1
0 0 1 0 0 0 0 0
0 1 1 0 0 0 0 0
#Output 3 0
1 0 0 0 0