TRABAJO FIN DE MÁSTER EN

SISTEMAS INTELIGENTES

CURSO 2010-2011

SISTEMA INTELIGENTE DE DETECCIÓN DE INTRUSIONES

Javier Martínez Puentes

Director:

Luis Javier García Villalba

Departamento de Ingeniería del Software e Inteligencia Artificial

---

MÁSTER EN INVESTIGACIÓN EN INFORMÁTICA

FACULTAD DE INFORMÁTICA

UNIVERSIDAD COMPLUTENSE DE MADRID

TRABAJO FIN DE MÁSTER EN

SISTEMAS INTELIGENTES

CURSO 2010-2011

SEPTIEMBRE 2011 – SOBRESALIENTE

SISTEMA INTELIGENTE DE DETECCIÓN DE INTRUSIONES

Javier Martínez Puentes

Director:

Luis Javier García Villalba

Departamento de Ingeniería del Software e Inteligencia Artificial

---

MÁSTER EN INVESTIGACIÓN EN INFORMÁTICA

FACULTAD DE INFORMÁTICA

UNIVERSIDAD COMPLUTENSE DE MADRID

Fdo:

Autorización de Difusión

Ea abajo firmante, matriculado en el Máster en Investigación en Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "***Sistema Inteligente de Detección de Intrusiones***", realizado durante el curso académico 2010-2011 bajo la dirección de Luis Javier García Villalba en el Departamento de Ingeniería del Software e Inteligencia Artificial, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

_____

Javier Martínez Puentes

Resumen

El sistema diseñado es un sistema de detección de intrusiones mediante el análisis del payload del tráfico de la red en busca de algún tipo de malware. Este sistema implementa su algoritmo de detección como "preprocesador dinámico" de Snort. Mediante el trabajo conjunto de Snort y del Sistema diseñado puede afirmarse que se obtiene un sistema altamente eficaz ante ataques conocidos (mediante el paso de reglas de Snort) e igualmente eficaz ante ataques nuevos o desconocidos (que era el objetivo prioritario del sistema diseñado).

Para resumir su funcionamiento bastaría decir que, como la mayoría de este tipo de sistemas, consta de dos fases: entrenamiento y detección.

Durante la fase de entrenamiento se crea un modelo estadístico del tráfico legítimo de la red mediante las técnicas Bloom Filters, N-Grams y Redes Neuronales (SOM y LVQ). Posteriormente se comparan los resultados obtenidos al analizar un dataset de ataques con dicho modelo de manera que se obtienen un conjunto de reglas que serán capaces de determinar si un determinado payload analizado corresponde a algún tipo de malware o por el contrario puede clasificarse como tráfico legítimo.

Durante la fase de detección se pasa el tráfico a analizar por el Bloomfilter creado en la fase de entrenamiento y se comparan los resultados obtenidos con las reglas que se produjeron durante la fase de entrenamiento.

Palabras clave

NIDS, Redes Neuronales, SOM, LVQ, BloomFilter, N-Gram, Payload, Malware, Snort.

Abstract

The designed system is an intrusion detection system by analysis of Payload from network traffic to look for some kind of Malware.

This system implements its detection algorithm as "dynamic pre-processor" of Snort. By working together of Snort and this system someone can affirm that a highly effective system to known attacks (by passing Snort rules) and equally effective against new and unknown attacks (which was the main objective of the designed system) is obtained.

To summarize the system working would suffice to say that, like most such systems, it consists of two phases: an initial training phase and a second phase of detection.

During the training phase a statistical model of legitimate network traffic through the techniques Bloom Filters, N-Grams and Neuronal Networks (SOM and LVQ) is created. Then the results obtained by analyzing a dataset of attacks with this model is compared and thus a set of rules that will be able to determine whether a payload analyzed corresponds to some kind of Malware or otherwise be classified as legitimate traffic is obtained.

During the detection phase the traffic for analyzing is passed by the Bloom Filter which it is created in the training phase and the obtained results with the rules that occurred during the training phase are compared.

Keywords

NIDS, Neuronal Networks, SOM, LVQ, BloomFilter, N-Gram, Payload, Malware, Snort.

Agradecimientos

Son muchas las personas a quienes debería nombrar en estas líneas, por su apoyo tanto a lo largo del desarrollo de este Proyecto Fin de Master como a lo largo de mi vida.

A mis padres y a mi hermana que siempre me han acompañado en los momentos en que más les he necesitado.

A todos mis amigos que han hecho más llevaderos estos años de estudio, que en verdad creo han sido los años más divertidos e interesante que he vivido.

A José Antonio López Orozco por su ayuda desinteresada con el sistema de redes neuronales desarrollado para este trabajo.

A Luis Javier García Villalba que siempre ha confiado en mis ideas y en mi trabajo. Gracias por la confianza y el apoyo que me ha brindado desde que comenzamos el Proyecto de Sistemas Informáticos.

Finalmente, me gustaría expresar mis más sinceros agradecimientos a todos aquellos que han contribuido a la ejecución de este trabajo directa o indirectamente.

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

Actualmente, cualquier experto en seguridad podría afirmar que uno de los principales problemas de seguridad de las redes es la proliferación de Malware, especialmente el aumento de Malware nuevo así como específico para alguna red determinada.

Podríamos nombrar algunas causas de este hecho: incremento del tamaño de las redes, aumento del valor de la información contenida en dichas redes, mejora tanto de las prestaciones de las conexiones a Internet como de la capacidad de procesamiento de los ordenadores conectadas a las mismas, etc. Sea cual sea la causa hay un dato indiscutible: cada día es mayor el número de Malware nuevo que surge [29].

Ante esta problemática han surgido diversos sistemas basados en reglas para detectar el envío de malware (así como para detectar otros tipos de ataques). Sin embargo, estos sistemas tienen el inconveniente de ir un paso por detrás de los desarrolladores de malware.

El resto de este capítulo está organizado como sigue: el apartado 1.1 presenta el objeto de investigación de este trabajo. Y en el apartado 1.2 se resume la estructura del resto del trabajo.

1.1. Objeto de la investigación

Diariamente salen a la luz nuevos especímenes que no comparten características con el malware ya conocido por lo que no existen reglas que sean capaces de identificarlo, lo que lo convierte en indetectable hasta que es demasiado tarde, puesto que lógicamente es necesario detectar previamente un malware dado para poder determinar algunas características del mismo a partir de las que se pudiese obtener alguna regla que lo identifique.

Por esta razón han surgido también diversos proyectos que se han propuesto evitar este inconveniente de manera que la detección del malware no dependa de un análisis detallado del mismo, sino que, por ejemplo, se base en la comparación con algún modelo construido o por el paso de unos determinados filtros para poder reconocerlos, de manera que se pueda detectar malware nuevo desde el primer momento de su propagación.

Visto el interés en este campo se decidió realizar un sistema de detección de malware. El Sistema diseñado se ha integrado dentro de Snort, de manera que fuese un "preprocesador dinámico" [4][42] de Snort.

A pesar de que dicho proceso fue inicialmente costoso debido principalmente a la escasez de documentación relativa puede afirmarse que los beneficios obtenidos han compensado dicho esfuerzo: con este tipo de diseño se puede aprovechar el proceso de re-ensamblado de paquetes (lo cual incrementa las prestaciones del sistema desarrollado), así como la detección por firmas que realiza Snort. Además, al poder integrarse dentro de Snort (uno de los NIDS con mayor difusión), la instalación y uso del preprocesador desarrollado es mucho más "amigable" para aquellos usuarios que ya han trabajado con Snort y a su vez su difusión puede ser mayor

1.2. Estructura del trabajo

El trabajo está organizado en 6 capítulos con la estructura que se comenta a continuación.

El Capítulo 2 realiza un repaso a las características generales de los IDS. Características utilizadas para calcular la eficiencia de este tipo de sistemas, arquitectura típica de los mismos, revisión del proceso completo de detección de intrusiones, repaso a los ataques típicos a los que hacen frente este tipo de sistemas y, por último, una clasificación de los mismos en función de diversas características.

El Capítulo 3 realiza un estado del arte de los principales IDS, analizando las características de este tipo de sistemas, aplicaciones y resultados que presentan.

El Capítulo 4 detalla las características, arquitectura y técnicas empleadas en de nuestro algoritmo.

El Capítulo 5 contiene los resultados obtenidos de las simulaciones realizadas del algoritmo.

Por último, el Capítulo 6 muestra las principales conclusiones extraídas de este trabajo.

2. VISIÓN GENERAL DE LOS IDS

Un IDS (Intrusion Detection System) es una herramienta de seguridad encargada de monitorizar los eventos de un sistema informático o red informática en busca de ataques que intenten comprometer la seguridad de dicho sistema.

Los IDS aportan a nuestro sistema una capacidad de prevención y de alerta anticipada ante cualquier actividad sospechosa. Y, aunque inicialmente no están diseñados para detener un ataque, sí pueden generar ciertos tipos de respuesta ante éstos.

Aumentan la seguridad de nuestro sistema. Vigilando el tráfico de nuestra red, analizando los paquetes recibidos en busca de datos sospechosos y en muchos casos detectando las primeras fases más comunes de los ataques (análisis de la tipología de la red, barrido de puertos, etc.).

2.1. Eficiencia de un IDS

Para medir la eficiencia de un IDS se utilizan una serie de características, que, a su vez, ayudan a comprender realmente qué es un IDS. Dichas características son: precisión, rendimiento, completitud, tolerancia a fallos y tiempo de respuesta.

2.1.1. Precisión

La precisión de un IDS es la capacidad que tiene el sistema para detectar ataques y distinguirlos del tráfico normal de una red.

Para medir la precisión de un IDS se utilizan dos ratios:

- **Porcentaje de falsos positivos**: Cantidad de alertas generadas al analizar tráfico legítimo con respecto al tráfico total de la red.

- **Porcentaje de falsos negativos**: Cantidad de ataques no detectados con respecto al tráfico de la red.

Lógicamente, cuanto menor sean estos ratios mejor podrá ser considerado nuestro detector. Sin embargo, en la práctica nos encontramos con la problemática de que cuando conseguimos llegar a un valor de falsos positivos que consideramos óptimo el porcentaje de falsos negativos aumenta a valores insatisfactorios, o el caso contrario, cuando obtenemos un porcentaje de falsos negativos satisfactorio, el porcentaje de falsos positivos es intolerable.

La fórmula de la precisión se convierte de esta manera en una herramienta con la que podamos comparar diversos IDS, o diversas configuraciones de un mismo IDS.

$$Presición = \frac{Ataques_{reales_{detectados}}}{Ataques_{reales_{detectados}} + Falsos_{positivos}} \quad (1)$$

Sin embargo, está fórmula no será siempre la más adecuada. Debemos tomarla únicamente como un punto de partida.

Imaginemos un sistema crítico en el que el nivel de seguridad ha de ser extremadamente alto. Para este caso en particular seguramente será mucho más conveniente penalizar los falsos positivos en favor de reducir lo máximo posible el porcentaje de falsos negativos.

El caso contrario es aquel en el que los medios disponibles para analizar las alertas son reducidos (en última instancia, en la mayoría de los sistemas, las alertas han de ser analizadas por técnicos especializados, por lo que este factor ha de ser tenido en cuenta). En este caso, no podemos generar un número elevado de falsos positivos, puesto que dado el caso, incluso las alertas que correspondiesen a ataques podrían pasar desapercibidas al tener que analizarse un número desorbitado de alertas. Dada esta situación, lo más conveniente sería penalizar el número de falsos negativos en favor de tener una cantidad de falsos

positivos manejable.

Llegados a este punto, tenemos que tener claro que la fórmula de la precisión ha de utilizarse únicamente como una referencia inicial y que, dependiendo de nuestras necesidades o medios disponibles, es conveniente utilizar modificaciones de dicha fórmula.

2.1.2. Rendimiento

El rendimiento de un IDS es el número de eventos que es capaz de analizar un sistema. Unas de las métricas más utilizadas para medir el rendimiento son el número de paquetes analizados por segundo, o más comúnmente, el numero de megabytes por segundo analizados.

Lo ideal en este caso es que nuestro IDS sea capaz de analizar el tráfico de la red en tiempo real, aunque en la práctica es inviable si se quiere realizar un análisis exhaustivo de la red.

Una vez más nos encontramos con multitud de posibilidades. Puede darse el caso de que nuestro sistema esté diseñado para ataques sencillos, por lo que a priori se puede suponer que el tiempo de análisis de cada paquete será reducido, por lo que podríamos tener un sistema con un rendimiento muy elevado.

Por el contrario, quizás deseamos diseñar un sistema que sea capaz de detectar todo tipo de ataques, desde los más simples a los ataques más complejos y elaborados. En este caso, aunque contemos con una gran cantidad de medios disponibles, lo más probable será que tengamos un rendimiento bajo, por lo que muchos paquetes no podrán analizarse, e irónicamente, aunque nuestro sistema tenga una capacidad de análisis muy elevado, mucho ataques pasarán desapercibidos simplemente porque no fueron analizados al ser el rendimiento de nuestro sistema demasiado bajo.

2.1.3. Completitud

Un IDS puede considerarse completo cuando es capaz de detectar todos los tipos existentes de ataques. Como parece lógico, construir un sistema que cumpla esta en condición en la práctica es inasumible, por lo que para diseñar nuestro IDS deberemos centrarnos en unos pocos tipos de ataques.

Debido a que los sistemas solamente suelen detectar unos pocos tipos de ataques distintos, en la práctica es necesario combinar diversas técnicas, o sistemas, para conseguir un grado de completitud aceptable.

Además, tenemos que tener en cuenta que la completitud es un factor que a su vez está relacionado con la precisión de nuestro sistema, es decir, cuanto más completo sea nuestro sistema menos preciso será, y análogamente, cuanto más preciso sea menor será el grado de completitud que podremos alcanzar.

Por ello nuevamente nos encontramos ante el reto de conseguir un equilibrio, esta vez entre la completitud y la precisión, aunque dependiendo de las necesidades que tenga que cubrir nuestro sistema podremos romper dicho equilibrio hacia un lado u otro.

La fórmula que determina la completitud de un sistema, la cual nos ayudará a obtener la configuración más adecuada de nuestro sistema como se muestra en (2).

$$Completitud = \frac{Ataques_{reales_{detectados}}}{Ataques_{reales_{detectados}} + Falsos_{negativos}} \quad (2)$$

Aunque como hemos comentado anteriormente, dicha fórmula no es una ley inamovible, y dependiendo de nuestras necesidades podremos modificarla en el sentido que más nos convenga.

2.1.4. Tolerancia a fallos

La tolerancia a fallos puede definirse como la capacidad de resistir los ataques. Un IDS para ser considerado robusto y eficaz, además de detectar la mayor cantidad de ataques generando el menor número de falsos positivos, ha de ser capaz de resistir en la medida de lo posible, que un ataque lo inutilice y, en consecuencia, deje a todo el sistema vulnerable frente a futuros (y más que probables después de un ataque de este tipo) ataques.

Además de resistir frente a los ataques que pueda sufrir el sistema, dentro de esta propiedad también se engloba la capacidad del IDS a resistir frente a un fallo en el sistema ya sea físico como un corte de luz o a nivel de software, quedando el sistema bajo el que corre el IDS bloqueado.

Consecuentemente, debido a los altercados que pueda sufrir un IDS, es conveniente que guarde tanto los patrones que utilice para la detección de los ataques como la configuración del mismo o los logs generados durante su ejecución. Así, ante algún imprevisto que inutilice nuestro IDS, la puesta a punto del sistema será lo más rápida posible y, a su vez, las alertas generadas antes de la interrupción podrán analizarse.

2.1.5. Tiempo de respuesta

El tiempo de respuesta de un IDS es el tiempo que tarda en reaccionar ante un ataque, ya sea lanzando una alerta, tomando las medidas necesarias para ralentizar dicha intrusión, o, en el mejor de los casos, abortarla.

Lógicamente, cuanto menor sea el tiempo de respuesta más efectivo podrá considerarse nuestro sistema, aunque debe asegurarse de que realmente existe un ataque antes de actuar, puesto que aunque es muy peligroso que nuestro IDS no detecte un ataque o lo detecte demasiado tarde poniendo en peligro nuestro sistema, en diversos casos puede ser igualmente peligroso que el IDS adopte medidas de contingencia sin haberse producido ataque alguno.

Al igual que ocurre con las anteriores características definidas de un IDS, podemos ver claramente como el tiempo de respuesta está estrechamente relacionado con la precisión del sistema o la completitud del mismo. Parece lógico pensar que cuanto más preciso sea nuestro sistema o completo ante distintos tipos de ataques que pueda detectar mayor será el tiempo de respuesta del mismo debido al análisis más exhaustivo que realizará del tráfico de la red. Nuevamente, según las necesidades del sistema a proteger debemos elegir unas prioridades a la hora de configurar o diseñar nuestro IDS.

2.1.6. Conclusiones

Como hemos visto hasta ahora, dada la complejidad de los IDS (y por supuesto de los ataques), antes de ponernos a diseñar un IDS (o configurarlo) debemos detectar las necesidades de nuestro sistema y el alcance del IDS encargado de protegerlo.

De esta manera, una vez que tengamos claras las necesidades del sistema, podremos dar prioridad a ciertas características anteriormente definidas en detrimento de otras.

Así, si estamos diseñando un IDS a medida de nuestro sistema, deberíamos realizar un diseño lo más flexible posible, para que en un futuro, en caso de modificarse las necesidades del sistema que protege, pudiéramos reutilizarlo modificando la configuración del mismo.

2.2. Arquitectura de un IDS

Los IDS, como veremos más adelante, pueden presentar características y objetivos muy distintos, por lo que cada sistema presentará su propia arquitectura.

Sin embargo, en este apartado presentaremos un esquema general de un IDS con los módulos más comunes, que servirá como patrón o punto de partida

para otros diseños más específicos.

En la figura 1 podemos ver el diseño típico de un IDS que a grandes rasgos estaría compuesto por una base de datos para almacenar la configuración del mismo y otra base de datos para almacenar el conocimiento, un módulo principal que es el encargado de la toma de decisiones y un módulo de respuesta ante los ataques.

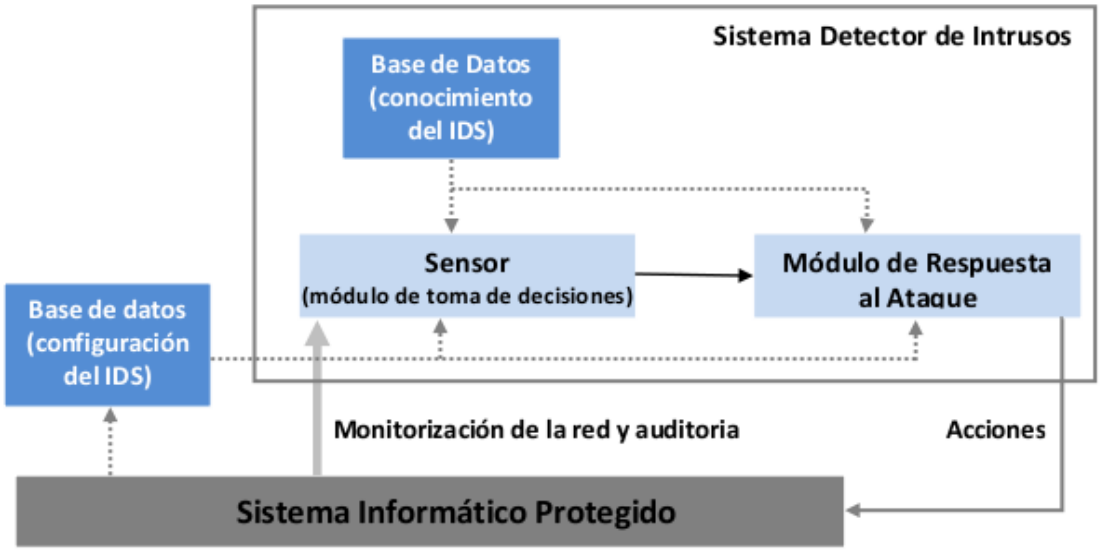


Fig. 1. Diseño típico de un IDS

A continuación detallaremos los módulos típicos del diseño de un IDS.

2.2.1. Sistema informático protegido

Este módulo representa al sistema informático que queremos proteger, aunque en la práctica el Sistema detector de Intrusos puede residir en el propio Sistema Informático Protegido.

Su función dentro del IDS consiste en permitir realizar cambios en la base de datos de configuración del IDS, ofrecer al IDS los datos que se desean analizar (ya sean datos que provienen de la monitorización de la red como de auditorías previas) y ofrecer una interfaz al usuario para poder lanzar la ejecución de nuestro IDS.

Finalmente señalar que el sistema informático será el módulo sobre el que se realizarán las acciones para evitar o responder ante ataques en el caso de que nuestro IDS solamente se encargue de la protección de dicho sistema informático. En el caso de que estemos protegiendo la totalidad de una red, el sistema informático será la herramienta que permita a nuestro IDS tomar las medidas de contingencia oportunas ante un ataque.

2.2.2. Base de datos de configuración del IDS

La base de datos de configuración de IDS se encargará de almacenar la configuración del mismo.

Aunque por defecto es común que los IDS se distribuyan con algún fichero de configuración, es recomendable que el administrador o usuario responsable del sistema a proteger, realice un sistema de ajuste mediante el fichero del configuración de manera que el IDS se adapte en la medida de lo posible a las necesidades o prioridades que considere oportunas el administrador.

Hay que destacar, que el proceso de configuración del IDS es de vital importancia, puesto que la diferencia entre un sistema correctamente protegido y otro vulnerable puede deberse a la poca atención dedicada a la configuración del mismo.

Los parámetros especificados en este módulo pueden ir desde la elección del modo de funcionamiento del IDS (por ejemplo, los modos de funcionamiento de Snort) hasta parámetros propios del funcionamiento interno de los detectores (por ejemplo el tipo de preprocesador usado o la activación de filtros para determinados protocolos).

Como se ve en el esquema, esta base de datos se comunica con el módulo de toma de decisiones (sensor) y con el módulo de respuesta al ataque.

2.2.3. Base de datos de conocimiento del IDS

La base de datos interna del IDS contiene los patrones de comportamiento que necesita el mismo para analizar los diferentes eventos del sistema informático y determinar si para dichos eventos han de generarse alerta o no.

Estos patrones de comportamiento serán muy diferentes dependiendo del tipo de IDS. Por ejemplo, algunos IDS pueden tener una base de conocimientos basado en un dataset de ataques. Por el contrario, para otros IDS la base de conocimiento estará compuesta por el tráfico normal o legítimo que suele tener la máquina protegida. En resumen, podemos definir la base de datos del conocimiento como aquel conjunto de datos que utilizará el IDS para generar alertas durante el análisis.

La información contenida en esta base de datos será usada por el módulo de toma de decisiones (sensor), a la hora de analizar el tráfico recibido y determinar si existe algún tipo de riesgo, y por el módulo de respuesta para establecer el protocolo de cuarentena o contingencia frente a algún tipo de ataque.

2.2.4. Sensor

El sensor, también llamado módulo de toma de decisiones, es el encargado de analizar los datos que son enviados al IDS y determinar con ayuda de la base de datos del conocimiento si dichos datos pueden considerarse una intrusión en el sistema de la que tenga que informarnos y/o tomar las contramedidas oportunas.

Este módulo utiliza la información de la base de datos de configuración para ajustar variables o parámetros internos del método que utilice para realizar el análisis del tráfico que recibe.

Una vez detectado un ataque (o generado un falso positivo), el sensor manda

una señal al módulo de respuesta del IDS. Lo más usual es que acompañe dicha señal con los datos que generaron la alerta, de manera que en un futuro ayude al administrador o responsable del sistema a obtener más información sobre el ataque, sirviendo como futuro entrenamiento del mismo o simplemente como una evidencia más del ataque para tomar las medidas legales que se consideren oportunas.

Debido a que el sensor es el módulo más importante de un IDS puede estar dividido a su vez en diversos submódulos. Por ejemplo, uno por cada algoritmo distinto de análisis (en el caso de que nuestro IDS esté compuesto por varios algoritmos distintos de detección trabajando de manera conjunta) o un módulo por cada etapa que componga nuestro algoritmo de detección (en el caso de sistemas de detección compuestos por algoritmos de detección muy sofisticados).

2.2.5. Módulo de respuesta a ataques

El módulo de respuesta se encarga de realizar las acciones pertinentes, previamente definidas, para cada tipo de ataque.

Cuando el sensor genera una alerta lo notifica inmediatamente a este módulo incluyendo generalmente los datos que generaron dicha alerta o en su defecto alguna otra información de relevancia que haya generado el sensor.

En este módulo, según la complejidad del IDS, además de registrar el ataque y los datos que hicieron saltar las alarmas (éstas serían las acciones básicas que deberían realizar todos los IDS) puede que el sistema incluya algún método de clasificación del ataque, de manera que junto con la información contenida en la base de datos de conocimiento y la base de datos de configuración se tomen las "contramedidas" oportunas frente al ataque.

El protocolo de actuación de un IDS frente a los ataques dependerán de la complejidad del módulo de respuesta y del tipo de ataque que hizo saltar las

alarmas. Dichas acciones van desde el registro de simples alarmas (logs) hasta la ejecución de ciertos comandos para intentar frenar el ataque o minimizar sus consecuencias e incluso en los IDS más complejos, se podría tomar alguna medida considerada "ofensiva" contra la fuente del ataque.

2.3. Proceso de Detección de Intrusiones

Como podemos ver en la figura 2, el proceso que lleva a cabo un IDS para detectar las intrusiones en el sistema que protege consta generalmente de cuatro fases: prevención, monitorización de la intrusión, detección de la intrusión y respuesta.

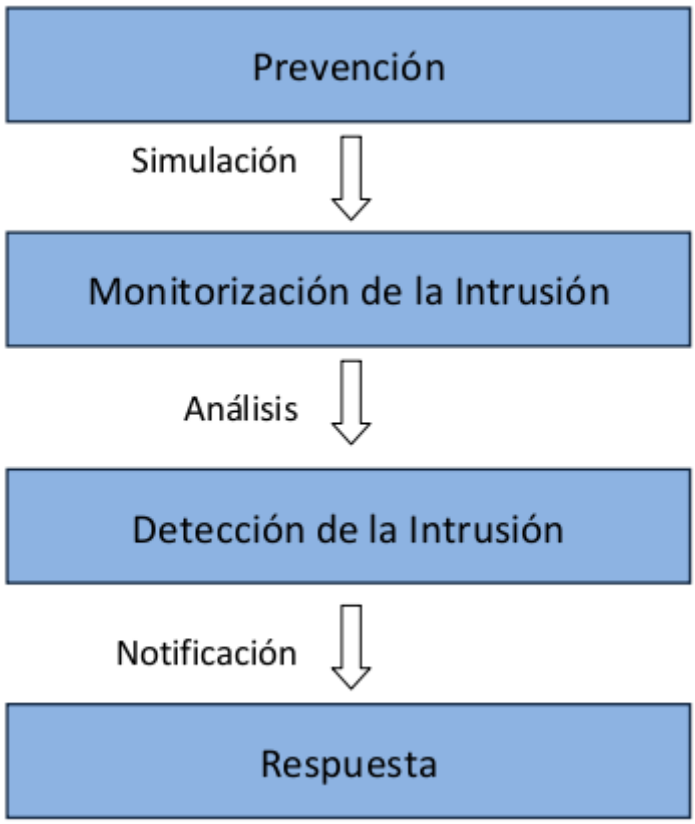


Fig. 2. Proceso que realiza un IDS para detectar las intrusiones en el sistema que protege

A continuación explicaremos cada una de estas fases.

2.3.1. Prevención

La función principal de un IDS es indudablemente la de evitar los ataques. Tareas secundarias son las relativas al registro de los ataques que se hayan producido o de las alertas generadas a partir de los mismos. Conviene reseñar, por tanto, que la principal función es la de evitar que los ataques al sistema tengan éxito.

La primera etapa de un IDS consiste en realizar una simulación con el tráfico que está circulando por la red o recibiendo el sistema a proteger y determinará si se puede llegar a una situación sospechosa. Si se produce ésta pasará a la siguiente fase del proceso de detección de intrusiones, la monitorización de la intrusión.

En resumen, podríamos decir que esta primera etapa consistiría en un primer filtro o análisis superficial del tráfico que determinaría si se ha de realizar un análisis más exhaustivo que, lógicamente, consumiría más recursos.

2.3.2. Monitorización de la Intrusión

Cuando el sistema detecte actividad sospechosa monitorizará el tráfico para que que pueda ser analizado, o lo que es lo mismo, tras un análisis superficial del tráfico, se pasará a realizar un análisis más detallado en los casos que se hayan clasificado como sospechosos en la etapa anterior.

Cabe destacar que muchos IDS comienzan su proceso de detección en esta etapa, y por lo tanto no realizan un filtrado previo del tráfico a analizar. Esta opción tiene sus ventajas y sus inconvenientes. Al realizar un análisis completo de todo el tráfico nos aseguramos de que no se pasen por alto ataques que "a simple vista" pasarían un primer filtro de análisis. El precio a pagar es que saturaremos en mayor medida el sistema. Como hemos comentado anteriormente, ninguna opción es siempre la mejor. Hay que encontrar un equilibrio, la configuración que mejor se adapte a las necesidades del sistema a

proteger o a los objetivos que nos hayamos fijado para el mismo.

2.3.3. Detección de la intrusión.

Después de analizar el tráfico el sistema ha de ser capaz de generar alertas o actualizar los logs del sistema en caso de que considere que se está sufriendo un ataque. Si tal hecho ocurre, enviará una señal al módulo de respuesta del IDS para que se tomen las medidas de protección necesarias.

2.3.4. Respuesta

Generalmente la respuesta de los IDS ante un ataque básicamente consiste en el registro de dicha alerta en el fichero de log´s correspondiente, incluyendo cierta información relevante de dicha alerta, o inclusive el tráfico que hizo saltar la alarma del sistema.

Sin embargo, los IDS más sofisticados pueden incluir algunas medidas de contingencia o cuarentena, como puede ser cerrar algún puerto, impedir que se procese más tráfico que provenga de la misma fuente que generó la alerta... e incluso, algunos IDS son capaces de tomar alguna medida "ofensiva" o "contramedidas" hacia la fuente del ataque.

Otra forma de explicar el proceso completo de un IDS para la detección de intrusiones es estudiar detalladamente la arquitectura del mismo.

El esquema de la figura 3 representa el flujo de los datos desde que son recogidos hasta que finalmente se determina si pueden formar parte de un ataque.

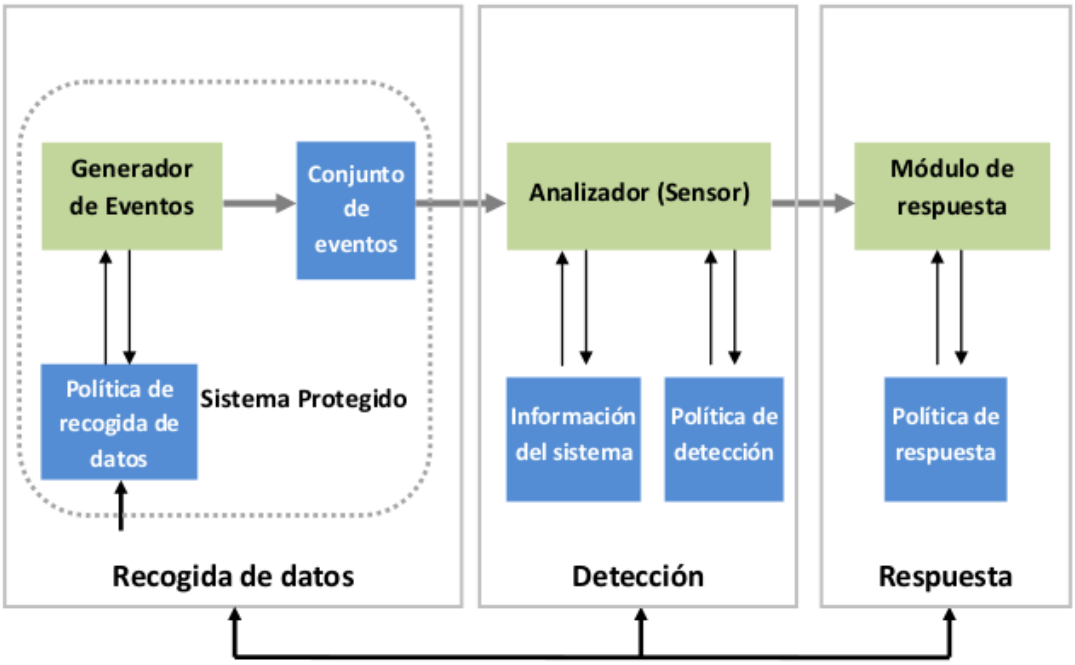


Fig. 3. Esquema del flujo de los datos

Como podemos ver, la estructura de un IDS puede dividirse en tres zonas claramente diferenciadas: recogida de datos, detección y respuesta.

En primer lugar, se recogen los datos siguiendo una política determinada. Esta política se refiere a la fuente y al momento en el que se recogen los datos. Por ejemplo, podría determinarse que solamente fuera objeto de análisis el tráfico que entra por determinados puertos (o al contrario, que el tráfico de determinados puertos no se analizase), o centrar el análisis en el tráfico recibido de diversas fuentes, en un intervalo horario determinado, según el tamaño del mismo,...

Una vez recogido el tráfico pasamos al generador de eventos, que generará diversos conjuntos de eventos en función de los tipos de datos recogidos (syslogs, paquetes de la red o información del sistema). En este punto debemos recalcar que un IDS no tiene por qué centrarse únicamente en el tráfico de la red. También existen IDS´s que analizan el comportamiento de los sistemas que protegen en busca de algún comportamiento anómalo que pudiera ser síntoma de estar sufriendo algún ataque.

Estos conjuntos de eventos se usan en el módulo de detección para determinar si puede existir un ataque al sistema. Como hemos dicho antes el sensor usa la información proporcionada por el sistema y una política determinada de detección para clasificar los datos proporcionados como ataques o como tráfico normal. Es en este módulo dónde se aplica el algoritmo de detección que determina si los datos analizados pueden suponer una evidencia de estar sufriendo un ataque o no.

Finalmente, los datos generados por el módulo de detección pasan al módulo de respuesta que seguirá una política definida para responder a los ataques detectados. Esta política podría consistir desde registrar únicamente la alerta, acompañando dicha alerta de algún tipo de información complementaria o incluso las fuentes originales que generaron dicha alerta, hasta tomar una serie de medidas activas de protección de nuestros sistemas e incluso "contramedidas" hacia la fuente del ataque.

2.4. Ataques

La finalidad de los IDS es detectar las intrusiones o ataques que sufre el sistema encargado del proteger. Para realizar dicho cometido los antivirus comerciales podrían parecer una buena solución. Sin embargo, como hemos mencionado anteriormente, cada día surgen nuevos ataques y vulnerabilidades cada vez más heterogéneas, hecho que complica la protección de un sistema con un simple antivirus, por lo cual surge la necesidad de desarrollar sistemas más complejos que sean capaces tanto de detectar ataques antiguos como los ataques más actuales y sofisticados.

A continuación mostraremos una sencilla clasificación de los ataques más comunes en función de diversos factores: según la naturaleza del ataque o según su origen. Posteriormente utilizaremos una clasificación más elaborada.

Según la naturaleza de los ataques estos pueden clasificarse en:

- **Pasivos**. Se trata de intrusiones en un sistema sin consecuencias para este. Por lo general se hacen para demostrar las vulnerabilidades de un sistema o como retos que se ponen a sí mismos los hackers más experimentados. El único propósito del ataque entrar en sistemas muy protegidos. Para demostrar el nivel de habilidad del atacante o la ineficiencia de las defensas del sistema.

- **Activos**. En este tipo de ataques la intrusión en el sistema se usa para dañarlo modificando, eliminando archivos o sustrayendo información confidencial. Son los ataques más perjudiciales y con consecuencias más graves. Según el nivel de confidencialidad de la información del sistema estos ataques pueden suponer un coste muy elevado para la compañía que sufre el ataque, por lo que estos sistemas deberían contar con los IDS más sofisticados y potentes.

Según el origen de la intrusión los ataques pueden ser:

- **Internos**. La fuente del ataque proviene de la propia red de la organización. Puede ser realizado por usuarios de la red o por atacantes que suplantan alguna identidad de usuarios con permisos sobre la misma. Son muy peligrosos si los privilegios que se obtienen sobre el sistema son muy elevados (especialmente si han suplantado la identidad de un administrador) por lo que frecuentemente comprometen la integridad de la información del sistema.

- **Externos**. Los ataques que provienen del exterior del sistema, en general se hacen a través de internet. Son los ataques más conocidos y lo que se conoce popularmente como hacking o pirateo informático. Aunque estos ataques son detectables con mayor facilidad que los ataques internos, una vez que el atacante tiene acceso a la red interna de la organización son tan peligrosos como los ataques internos.

A continuación se muestra una breve clasificación sobre los ataques más frecuentes y sus posibles consecuencias en el sistema en caso de éxito.

2.4.1. Monitorización

Estos ataques se ejecutan para observar a la víctima y su sistema. De esta manera, es posible obtener información muy valiosa con el objetivo de enumerar sus debilidades y encontrar posibles formas de acceso al sistema monitorizado. También podemos obtener información valiosa sobre el comportamiento típico de los usuarios de un sistema, de manera que nuestro ataque se camufle dentro de dicho comportamiento para pasar desapercibido.

2.4.1.1. Decoy

Son programas que imitan la interfaz de otras aplicaciones usadas normalmente por los usuarios. Estos programas suelen requerir la identificación del usuario mediante su login y contraseña, de manera que una vez que el usuario los ejecuta, el atacante obtiene el usuario y contraseña de su víctima. Posteriormente, el programa guarda la información en un fichero y ejecuta la aplicación a la que imitan, de manera que el usuario no detecte nada anormal. El atacante podrá así suplantar la identidad del usuario del sistema y acceder a cualquier recurso al que pudiera acceder el usuario legítimo.

Otros ataques de este tipo son los *Keyloggers* que guardan todas las pulsaciones de teclado que realiza el usuario en su sesión. De esta manera, el atacante puede acceder en cualquier momento al fichero generado para obtener contraseñas y nombres de usuarios.

2.4.1.2. Escaneo

El escaneo es el método mediante el cual el atacante descubre canales de comunicación susceptibles de intrusión debido a diversos fallos de seguridad.

Consiste en el envío de paquetes mediante diferentes protocolos a los puertos de la máquina víctima para averiguar qué servicios (puertos) están abiertos observando el comportamiento de los mismos (recepción o extravío de paquetes de respuesta).

Existen gran variedad de tipos de escaneo puesto que con el tiempo la detección de los métodos más frecuentes de escaneo han sido incorporadas a los IDS, por lo que han ido surgiendo nuevos métodos de escaneo según se estandarizaba la detección automática de técnicas antiguas u obsoletas.

A continuación enumeramos los tipos de escaneo más frecuentes según las técnicas, puertos objetivos y protocolos utilizados:

- **TCP connect**. Forma básica de escaneo de puertos. Se intenta establecer conexión con varios puertos de la máquina objetivo. Si el puerto está escuchando, devolverá una respuesta de éxito en la conexión al inicializar la misma. Cualquier otro evento significará que el puerto está cerrado.

  Esta técnica se caracteriza por ser rápida y no precisar ningún permiso de usuario sobre la máquina víctima. Sin embargo, es una técnica que se detecta fácilmente ya que los intentos de conexión queda registrados en la máquina objetivo, generando lo que comúnmente se conoce como "ruido".

- **TCP SYN**. Este escaneo usa la técnica de "la media apertura". Consiste en mandar un paquete TCP SYN al puerto objetivo. Si este puerto está abierto contestará con un paquete ACK y si no es así responderá con un paquete RST. Si el puerto está abierto se responderá al paquete ACK enviado por la víctima con un paquete RST para no establecer la conexión y no dejar rastro en la máquina objetivo.

  Es una técnica poco ruidosa y muy sutil ya que no se llega a establecer conexión en ningún momento.

- **TCP FIN**. Consiste en el envío de un paquete FIN a un puerto. Si éste responde con un RST el puerto estará cerrado. Sin embargo, si no se obtiene ninguna respuesta significa que el puerto está abierto.

  Esta técnica es la más efectiva para no ser detectada, sin embargo, sólo funciona en sistemas LINUX/UNIX ya que en Windows siempre se responde con un RST a los paquetes FIN.

- **Escaneo fragmentado**. Este procedimiento consiste en fragmentar los paquetes de sondeo dirigidos a la máquina víctima. Con esto conseguimos provocar menos ruido en las herramientas de protección del sistema objetivo. Lógicamente, este tipo de escaneo puede combinarse con algún escaneo de los comentados anteriormente o algún otro.

2.4.1.3. Sniffering

Consiste, una vez dentro de una red de ordenadores, en ejecutar en alguna de las máquinas del sistema un programa espía llamado sniffer. Este programa registra en un fichero todo el tráfico que pasa por la red o parte de él (generando menos carga de trabajo en la víctima y por lo tanto llamando menos la atención de los IDS que protegen la red).

En este fichero pueden encontrarse todo tipo de datos que circulen por la red como contraseñas, números de cuenta bancaria, información confidencial de valor, etc. Por lo general, estos datos estarán cifradas por lo que su lectura no siempre será fácil.

2.4.2. Validación

Este tipo de técnicas tienen como objetivo suplantar la identidad de un usuario con acceso a la red mediante el uso de sus cuentas de acceso y contraseñas.

2.4.2.1. Spoofing – Looping

Consiste en acceder a una máquina del sistema (con una contraseña que ya tenemos) y obtener información de la red y del resto de equipos suplantando nuevas identidades de usuarios de ordenador en ordenador (siempre y cuando estén conectados a la red).

Por ejemplo, si estamos en el salto 4 y realizamos un ataque de escaneo a otra máquina se registrará como autor del ataque al usuario del ordenador 4. Si realizamos este paso con cierta frecuencia, es posible que mientras que los administradores de la red consiguen cerrar el acceso a la misma desde ordenadores atacados hace n pasos, el atacante todavía disponga de los sistemas atacados en los n-1 pasos para continuar con su actividad.

Además, estos ataques son muy útiles para ocultar la identidad del intruso ya que mediante una investigación para detectar el origen del ataque, debe pasarse por todas y cada una de las máquinas que fueron atacadas hasta llegar al origen de la intrusión.

2.4.2.2. Suplantación de IP

Consiste en generar paquetes de información con una dirección IP falsa. Para ello cambiamos la cabecera del paquete asignándole una IP de origen de algún usuario del sistema. De esta forma se consigue culpar a otra persona del ataque, incluso que la red de la que proviene el ataque es otra.

2.4.2.3. Suplantación DNS

Consiste en la manipulación de paquetes del protocolo UDP para engañar al servidor de dominio y que éste nos entregue información de sus tablas de registros. Esto es posible sólo si el servidor DNS funciona en modo recursivo,

modo por defecto en los servidores de dominio.

2.4.2.4. IP Splicing-Hijacking

Consiste en interceptar una conexión establecida recientemente entre un usuario y el servidor (ordenador víctima). En un momento dado se envía al servidor un paquete casi idéntico al que espera recibir del usuario. Así conseguimos suplantar su identidad. En ese momento el intruso recibe la respuesta del servidor y puede seguir mandando paquetes, ya que el servidor cree que es el usuario el que los envía. El usuario inicial se queda esperando datos como si su conexión se hubiera colgado.

2.4.2.5. Puertas traseras

Las puertas traseras son programas o funcionalidades ocultas dentro de un programa o sistema que se usan para acceder a ellos sin pasar los procedimientos habituales de validación.

La puertas traseras suelen utilizarse para facilitar el trabajo a los desarrolladores durante las pruebas del sistema durante la implantación o desarrollo del mismo. El problema reside en que en ocasiones se olvida la existencia de dichas puertas traseras, por lo que siguen pudiendo utilizarse después de la implantación de los sistemas, suponiendo una importante brecha de seguridad en los mismos.

Si un atacante descubre una puerta trasera o consigue ejecutarla en la máquina objetivo podrá entrar en ella sin necesidad de identificarse y, en la mayoría de los casos, obtendrá un control total del sistema.

2.4.2.6. Uso de exploits

Los exploits son programas que aprovechan las vulnerabilidades conocidas de los sistemas operativos para dar al intruso la identidad de súper usuario o

algún otro privilegio en la máquina víctima. Con estos privilegios el atacante puede conseguir el control prácticamente total del sistema y acceder a los datos residentes o a funcionalidades del mismo.

2.4.3. Denegación de Sevicios (DOS)

Este tipo de ataques pretenden desbordar los recursos de la máquina objetivo de forma que se interrumpen los servicios que suministra. También pueden centrarse en saturar la red en la que reside la víctima y a través de la cual ofrece determinados servicios. El objetivo es bloquear estos servicios ofrecidos por el ordenador a sus clientes.

2.4.3.1. Flooding

Este tipo de ataque desborda los recursos del sistema. Consiste en saturar al ordenador víctima con mensajes que requieren establecer conexión y dar respuesta.

El ataque consiste en enviar mensajes a la víctima con una dirección IP falsa, de esta manera, la víctima intenta dar respuesta a las solicitudes, pero como la dirección IP es falsa, no obtiene una respuesta satisfactoria al enviar la solicitud, por lo que procede a reenviarla.

Finalmente, si el ataque tiene éxito, el buffer de la víctima se acaba saturando con información de conexiones abiertas en espera de respuesta, por lo que se impide a la víctima que pueda procesar otras solicitudes de usuarios legítimos.

Un ataque de este tipo muy famoso es el llamado "ping de la muerte" ante el cual ya existen numerosas medidas de protección implementadas en prácticamente todos los sistemas. Pero que, en su momento, causó auténticos estragos en los administradores de sistemas.

2.4.3.2. SYN Flood

Consiste en mandar paquetes SYN (paquetes para iniciar el establecimiento de una conexión) a la víctima y no contestar a los paquetes ACK (paquetes enviados por la víctima aceptando la conexión) colapsando la pila TCP/IP de la víctima a la espera de recibir la repuesta definitiva para establecer la conexión por parte del atacante.

Para que este ataque sea óptimo debemos mandar los paquetes con una IP inexistente. De esta manera la víctima no podrá responder ni finalizar la conexión hasta que expire el tiempo máximo de espera establecido en la víctima y se desestime finalmente el intento de conexión. Si el atacante genera miles de estas llamadas la máquina objetivo se colapsará.

2.4.3.3. Connection Flood

La clave de este tipo de ataque es la sencillez del mismo. Consiste en saturar los sistemas de la víctima enviándo más peticiones de conexión de las que puede soportar.

Así conseguiremos que otros usuarios, aquellos que quieren acceder al sistema para fines legítimos, no puedan acceder al servicio que ofrece. Aunque a medida que pasa el tiempo estas conexiones se van liberando, al atacante simplemente le basta con seguir mandando peticiones, a un ritmo mayor al que se procesan, para mantener la máquina saturada.

2.4.3.4. Ataque Land

Consiste en mandar paquetes a un puerto abierto de la máquina objetivo con la misma dirección y puerto origen que la dirección y puerto destino. Tras varios envíos de estos paquetes la máquina deja de responder. Esta técnica sólo funciona en sistemas Windows.

2.4.3.5. Tormenta Broadcast

Consiste en mandar una petición ICMP a la dirección broadcast de la red de la víctima. Para ello únicamente debemos enviar una petición ICMP indicando como dirección IP origen la dirección IP del sistema que queremos atacar.

El dispositivo broadcast lanzará esta petición a todos los terminales de la red que responderán simultáneamente a la máquina objetivo. De esta manera, "utilizando" los propios sistemas de la víctima, que responderá a la petición simultáneamente, la saturaremos intentando enviar una cantidad de respuestas superior a su capacidad de procesamiento, produciendo un colapso en su sistema.

2.4.3.6. Desbordamiento NET

Consiste en lanzar a la red a la que se quiere atacar una gran cantidad de paquetes sin sentido que colapsen el tráfico que circula por la red. De esta forma ralentizamos o paramos el tráfico legítimo que debería poder atravesar la red y llegar a sus destinatarios legítimos.

2.4.3.7. Obb o Supernuke

Consiste en enviar paquetes UDP modificados, con la señal Out of Band activada a los puertos de escucha del NETBIOS (137-139). La máquina los interpreta como no válidos, pasa a ser inestable y se cuelga. Sólo funciona en determinados sistemas Windows aunque actualmente existen parches que protegen a las máquinas de este ataque.

2.4.3.8. Tear drop One y Tear drop Two

Este ataque consiste en enviar paquetes fragmentados con datos que se solapen entre sí. Algunas implementaciones de colas IP no pueden ensamblar correctamente este tipo de paquetes, por lo que se acaba saturando la pila y

finalmente el ordenador se cuelga.

Por ejemplo, enviar un paquete con los datos del 1 al 100 y después enviar otro con los datos del 50 al 150 produciría el mal funcionamiento de la máquina objetivo. Este ataque funciona en diversas versiones de Unix, Linux y Windows.

2.4.4. Modificación

Este tipo de técnicas tienen como objetivo la modificación no autorizada de los datos y ficheros de un sistema. También pueden modificarse programas que se ejecutan en el sistema cambiando su funcionamiento o el tráfico que reciben o envían dichos sistemas a través de la red.

2.4.4.1. Tampering

Estos ataques están relacionados con la modificación sin autorización de datos o de los programas instalados en el sistema, incluido el borrado de ficheros. También pueden modificar el contenido de datos confidenciales o valiosos para la estrategia de negocio de la víctima.

Son especialmente peligrosos cuando el atacante tiene permisos de súper usuario ya que podrá borrar cualquier información del sistema y dejarlo inservible.

2.4.4.2. Borrado de huellas

Este tipo de ataque puede considerarse un complemento de otros tipos más dañinos. Consiste en modificar los ficheros log del sistema operativo de la máquina asaltada, donde queda constancia de la intrusión para borrar el rastro dejado por el atacante.

Cuando un intruso abandona una máquina debe borrar todo rastro de la sesión iniciada y los movimientos realizados. Esto permite no despertar sospechas en los administradores del sistema y poder realizar nuevos ataques

aprovechando la misma vulnerabilidad.

De esta manera se dificulta el rastreado del origen del ataque y, lo que es más peligroso, la metodología seguida o brecha en la seguridad aprovechada para lograr su objetivo.

2.4.4.3. Ataques Java Applets

Los Applets de Java son ficheros ejecutables de programas y, en consecuencia, pueden ser modificados para que realicen acciones específicas de ataque. Por ello, Java implementa grandes medidas de seguridad por lo que es difícil realizar estos ataques.

2.4.4.4. Ataques Java script o Visual script

Los Java script o Visual script son capaces de iniciar aplicaciones en la máquina cliente o víctima para grabar información del disco duro, con lo cual resultan útiles para insertar virus u otros programas maliciosos en el ordenador del usuario.

2.4.4.5. Ataques ActiveX

Los controles ActiveX son necesarios para la visualización de muchas páginas web. Si el usuario no dispone de ellos piden autorización para ser instalados en el sistema. Un control ActiveX es legítimo si viene acompañados de un certificado de autenticidad. Un atacante puede falsificar uno de estos certificados o, simplemente engañar al usuario, consiguiendo de esta manera que la víctima ejecute un programa determinado.

Un ejemplo conocido es un control ActiveX que desactivaba la petición de aceptación de los controles ActiveX de los navegadores cuando se ejecutaba. De esta manera, el ordenador víctima aceptaba la ejecución de todos los controles que encontraba en su navegación asumiendo que el usuario había aceptado su

instalación.

2.5. Clasificación de los IDS´s

Los IDS se pueden clasificar de varias formas atendiendo a una serie de características. En el esquema presentado en la figura 4 se pueden observar estos criterios y los principales tipos de IDS.

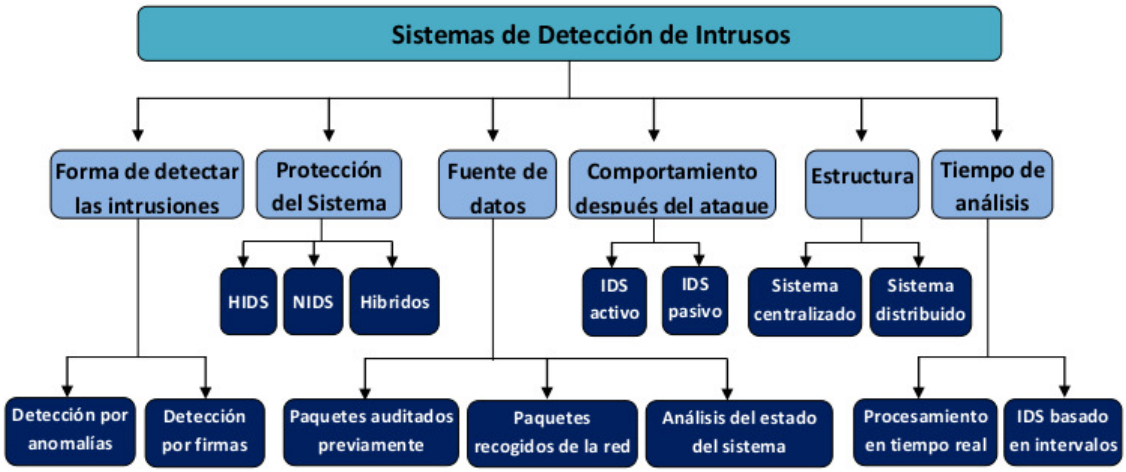


Fig. 4. Clasificación de los IDS´s

A continuación vamos a explicar los distintos criterios utilizados para dicha clasificación, los tipos de IDS encuadrados en cada cada categoría y las ventajas que tienen unos respecto de otros.

2.5.1. Clasificación respecto a la forma de detectar las intrusiones

Esta clasificación se basa en la forma que tiene el IDS de detectar un ataque al sistema, Esto es, en cómo procesa el tráfico recogido para determinar que hay una intrusión. Tenemos dos tipos:

2.5.1.1. Detección por anomalías

Trata de detectar intrusiones estudiando los usos del sistema en busca de situaciones anómalas. Para determinar si una determinada situación puede considerarse anómala o no el IDS crea perfiles de conducta, de manera que todo

comportamiento que se sale de los perfiles más comunes es considerado como un posible ataque.

El sistema tiene que ser entrenado para crear los perfiles que determinen el comportamiento normal en el sistema. Esta fase de entrenamiento debe realizarse con tráfico limpio (sin ningún tipo de ataques) para garantizar que el perfilado del sistema es correcto. El principal problema radica en que los perfiles de comportamiento no son estáticos: pueden ir evolucionando para adecuarse a los nuevos usos del sistema protegido.

La principal ventaja de estos sistemas es que pueden detectar ataques nuevos basándose en el comportamiento anómalo que ocasionen éstos en el sistema y que cuentan con un alto porcentaje de detección de ataques.

Las desventajas de este sistema son el alto número de falsos positivos que generan ya que cualquier actividad nueva en el sistema que difiera de manera significativa con las "conductas habituales" se considerará como un ataque.

La principal vulnerabilidad de este tipo de IDS es que se produzca un ataque durante la fase de entrenamiento ya que éste se considerará en un futuro como tráfico legítimo. Además, debido a la característica de que los perfiles van evolucionando, un intruso que conozca el sistema puede entrenarlo para que no detecte futuros ataques.

Los IDS basados en anomalías son, por lo general, muy caros computacionalmente.

2.5.1.2. Detección por firmas

Estos sistemas contienen un conjunto de reglas que usarán para detectar los ataques. Su funcionamiento consiste en recoger el tráfico, analizarlo y ver si se corresponde con alguna regla en cuyo caso se identificará dicho tráfico como una intrusión y se actuará en consecuencia.

Este tipo de IDS está muy extendido debido a la facilidad de su uso y a la gran cantidad de alternativas desarrolladas que existen (entre ellas Snort). Los conjuntos de reglas forman la base del módulo de detección y de ellas dependerá la eficiencia del sistema. Por lo general, el inconveniente que presentan este tipo de sistemas es que han de configurarse correctamente en función de la red a la que tengan que proteger con el fin de sacar todo el rendimiento al IDS.

La ventaja principal de estos sistemas es el bajo número de falsos positivos debido a que las reglas suelen ser bastante específicas. Además, pueden crearse fácilmente reglas para ataques nuevos identificados y así actualizar el detector. Igualmente, existen comunidades de administradores de sistemas que actualizan constantemente las reglas según surgen nuevos ataques.Por último, destacar el mínimo coste computacional que tienen estos sistemas, lo que permite respuestas más rápidas a los ataques.

En cuanto a las desventajas de este tipo de IDS, la principal es que son pocos eficientes ante nuevos ataques, ya que si no existe una regla que identifique un ataque este no se detectará. Un problema cada vez mayor debido al gran número de ataques nuevos que surgen cada día. Esto ocasiona un gran esfuerzo para mantener el sistema actualizado mediante nuevas reglas que identifiquen estos ataques.

Por otro lado, estos sistemas pueden ser vulnerables si el atacante conoce las reglas del mismo (muchas veces de dominio público) y encuentra la forma de evitarlas. Por último decir que estos sistemas tienen problemas para detectar ataques internos al no tener en cuenta entre sus reglas las acciones desde dentro del sistema protegido.

2.5.2. Clasificación respecto al tipo de sistema protegido

Esta clasificación se basa en el lugar donde el IDS analizará el tráfico, es decir, el

tipo de protección que daremos al sistema.

2.5.2.1. HIDS (Host – based IDS)

Los HIDS están diseñados para monitorear, analizar y dar respuesta a la actividad de un determinado terminal (host). Su principal característica es que sólo protegen el terminal en el que se ejecutan.

El funcionamiento de estos sistemas consiste en monitorizar todo tipo de actividades que sucedan en un terminal como por ejemplo: intentos de conexión, intentos de log-in, movimientos de los administradores (que tienen más privilegios y pueden ser más peligrosos), la integridad del sistema de ficheros, etc.

La principal ventaja de estos sistemas es que son los más indicados para detectar ataques internos debido a su capacidad para monitorear las acciones y accesos al sistema de ficheros que un usuario realiza en un terminal. No hay que pasar por alto esta característica puesto que está demostrado que la mayoría de ataques a las empresas son realizados desde en interior de las mismas, de ahí la importancia de estos sistemas de detección.

Aún así, hay que señalar que el principal inconveniente que presentan estos IDS reside en que sólo protegen la máquina en la que se ejecutan por lo que su uso es muy específico. Para conseguir la protección de una red es necesario usar otro tipo de sistemas.

2.5.2.2. NIDS (Network IDS)

Los NIDS recogen y analizan todos los paquetes que circulan por la red que tienen que proteger. Esta captura de paquetes se puede realizar desde cualquier medio y con cualquier protocolo, aunque por lo general el tráfico es TCP/IP.

Una vez recogido el tráfico se analiza de formas muy distintas para intentar

detectar las intrusiones. Debido a que la captura se realiza antes de llegar a los terminales, estos sistemas suelen usarse como un primer perímetro de defensa en la protección de una red.

Las ventajas de estos IDS son que detectan los ataques antes de que lleguen a los terminales permitiendo una respuesta más rápida. Además, como hemos dicho, protegen toda la red en la que están instalados y no sólo un terminal como ocurre con los HIDS.

La desventaja de este tipo de IDS es la alta capacidad computacional que se necesitan para procesar detalladamente todo el tráfico que pasa por una red. Por ello, la práctica habitual es que el análisis de los paquetes sea menos exhaustivo que con los HIDS. Otra desventaja a tener en cuenta es que este tipo de IDSs suelen tener problemas con el tráfico cifrado, aunque este problema ya se está solucionando en los sistemas más recientes.

2.5.2.3. IDS Híbrido o NNIDS (Network-Node IDS)

Los IDS híbridos combinan los HIDS y los NIDS para proporcionar las ventajas de los dos sistemas y evitar en la medida de lo posible los principales inconvenientes que presentan. La mayoría de sistemas informáticos y redes importantes usan este tipo de detector.

Estos sistemas contienen dos módulos: uno analizará el tráfico que pasa por la red y otro, residente en los terminales, que analiza las actividades de los mismos. El sistema híbrido se encarga de gestionar las alertas de los dos módulos para identificar una intrusión en el sistema.

Como hemos dicho los sistemas híbridos tienen las ventajas de los HIDS y de los NIDS. La única desventaja que presentan es el mayor coste computacional y la dificultad para correlacionar correctamente las alertas de los dos módulos.

Dentro de los IDS híbridos tenemos los NNIDS. Estos sistemas también

recogen el tráfico de la red pero después de que éste haya sido dirigido hacia su terminal, es decir, analiza el tráfico de un solo terminal pero antes de que llegue a la máquina. La ventaja de estos sistemas es que pueden detectar el ataque antes de que los paquetes lleguen al terminal. Normalmente se utilizan como parte de un HIDS en los terminales críticos cuya seguridad es más importante para la organización y no es prioritario optimizar el coste en recursos.

2.5.3. Clasificación respecto a la fuente de datos

Otra forma de clasificar los IDS atiende a la procedencia de los datos que se van a analizar para detectar las intrusiones.

2.5.3.1. Paquetes auditados previamente

En estos IDS los paquetes provienen de un proceso de monitorización realizado previamente. Los datos están guardados en los dispositivos de almacenamiento y se analizarán después de que haya habido un ataque exitoso en el sistema. El objetivo de dichos análisis es encontrar al autor del ataque y/o la vulnerabilidad que ha aprovechado para acceder al sistema. Este proceso es conocido como análisis forense del tráfico de un sistema.

La característica principal de estos IDS es que analizan los paquetes de forma mucho más detallada y exhaustiva que otros sistemas. Debido a que no tienen que generarse alertas lo más rápido posible para evitar los ataques, puesto que ya han tenido lugar, se pueden recoger todos los datos posibles sobre los mismos. Sin embargo, aunque en este tipo de sistemas no prime la inmediatez en las respuestas, sí se valora la rapidez del sistema para detectar el problema de seguridad y así poder cerrar esa vía de acceso evitando futuros ataques.

2.5.3.2. Paquetes recogidos en la red

Este tipo de sistemas capturan el tráfico de la red actuando como un sniffer, y

una vez capturado el tráfico, el sistema lo analiza para detectar posibles ataques.

Esta forma de recoger los paquetes es usada habitualmente por los NIDS. Las ventajas, como hemos visto anteriormente, son que el tráfico puede procesarse antes de llegar a su destino y, en caso de detectar una intrusión, tener un tiempo de respuesta ante dicho ataque menor.

El punto negativo es que si la cantidad de tráfico es grande, el coste computacional también lo será, mermando el rendimiento del IDS.

2.5.3.3. Análisis del estado del sistema

En este tipo de IDS no se analiza tráfico de la red sino los datos que provienen del propio sistema (kernel, servicios, archivos). De esta manera se pueden detectar ciertos comportamientos sospechosos dentro del terminal, como por ejemplo aquellos que puedan considerarse estadísticamente poco frecuentes o aquellos que compartan características con ciertos patrones identificados como potenciales ataques.

Por ejemplo, podría considerar un ataque el uso de unos archivos determinados o el acceso al sistema de un usuario desde un terminal distinto al habitual.

Como hemos visto antes, este tipo de datos son usados por los HIDS. Las ventajas que presentan este tipo de sistemas son la detección de ataques internos y el poco coste computacional que supone el procesamiento de este tipo de datos. Por contra, presenta el problema de que sólo se protege el terminal del que analizan los datos.

2.5.4. Clasificación respecto al tipo de respuesta al ataque

Los IDS también se pueden clasificar según el comportamiento que presentan una vez que han detectado un ataque.

2.5.4.1. IDS activo

Los IDS activos, una vez detectado un ataque, generan la alarma correspondiente e intentan responder a dicho ataque para contrarrestarlo o minimizar los daños que pueda ocasionar.

Algunas de las acciones que realizan estos sistemas pasarán por cerrar puertos, aislar ciertos terminales, bloquear procesos, etc.

La ventaja de estos sistemas es que tratan de contrarrestar el ataque automáticamente sin que el administrador del sistema tenga que analizar la alarma y actuar en consecuencia como pasa con otro tipo de sistemas.

En el lado negativo tenemos que este tipo de sistemas presentan un gran coste computacional. Al tener que responder a los ataques, estos tienen que identificarse de forma más concreta para que las contramedidas que realicen sean las adecuadas.

Por otro lado, la dificultad de responder a un ataque de forma automática y efectiva es muy grande. Por ello, por lo general, estos sistemas realizan acciones de prevención que no pueden tener graves consecuencias en caso de tratarse de falsas alarmas.

2.5.4.2. IDS pasivos

Estos IDS se limitan a generar una alerta cuando detectan un ataque. Por lo general, estas alertas contienen información sobre los paquetes que han hecho saltar una alarma y el motivo por el cual son sospechosos.

Los IDS pasivos son más fáciles de implementar y por lo general tienen menor coste computacional. Sin embargo, requieren la supervisión de un técnico que analice las alertas y actúe en consecuencia para evitar las intrusiones o mejorar la seguridad del sistema.

Es necesario que el sistema sea preciso a la hora de lanzar las alertas ya que éstas tienen que ser revisadas. Así, deben evitarse en la medida de lo posible los falsos positivos, puesto que un número elevado de éstos supondría un coste demasiado elevado en medios humanos o directamente dejaría de analizar una gran cantidad de alertas.

2.5.5. Clasificación respecto al tipo de estructura del sistema

Esta clasificación se basa en el tipo de estructura hardware en la que se ejecuta el IDS.

2.5.5.1. Sistema centralizado

Los IDS con una estructura centralizada se caracterizan por ejecutarse sólo en un terminal, desde el cual se controlará toda la seguridad de una red.

La principal ventaja de estos sistemas es la facilidad que presentan para correlacionar eventos de distintas fuentes ya que todos los datos están en el terminal donde se ejecuta el IDS. Además, la instalación, configuración y mantenimiento es más fácil y menos costosa.

La desventaja de este tipo de IDS es la menor capacidad de cómputo con respecto a los sistemas distribuidos que aprovechan el potencial de varias máquinas para procesar los paquetes. Estos sistemas se ven limitados por la capacidad del terminal en el que se ejecuta que deberá ser grande para que el IDS tenga un buen rendimiento. Otro inconveniente es que si falla el terminal que ejecuta el detector la red quedará desprotegida.

2.5.5.2. Sistema distribuido

Estos sistemas se caracterizan por ejecutarse en varios terminales a la vez, ya sea por necesidad o por motivos de rendimiento. Este tipo de estructura se utiliza en los sistemas que usan HIDS ya que habrá que ejecutar el IDS en cada terminal que se quiera proteger. Por otro lado, este tipo de sistemas permiten

aprovechar la capacidad computacional de varias máquinas a la hora de analizar el tráfico recogido de una red.

Para implantar un sistema distribuido en la red se requiere un terminal que actúe como coordinador y recoja todas las alertas generadas por los terminales satélite, las correlacione y finalmente determine si se ha producido un ataque en el sistema.

La principal ventaja de este sistema es la mayor capacidad computacional que tiene para analizar los datos. En caso de tratarse de un HIDS esta distribución es necesaria si se quiere proteger más de un terminal.

La dificultad a la hora de recoger y relacionar correctamente todos los eventos de las distintas máquinas hace que la implementación del coordinador sea complicada. Además, aumenta el tiempo para detectar un ataque debido a los flujos de entrada/salida que se requieren para ello.

2.5.6. Clasificación respecto al tiempo de análisis

Esta clasificación se basa en la frecuencia en la que el sistema analizará el tráfico, es decir, si lo hace en tiempo real o según unos ciertos intervalos.

2.5.6.1. Procesamiento en tiempo real

En estos sistemas el análisis de los datos en busca de comportamientos sospechosos se hace en tiempo real. Por lo general, este análisis de los paquetes es bastante superficial (por ejemplo, analizar sólo las cabeceras de los paquetes).

Para realizar un análisis más exhaustivo se tendría que reducir la capacidad de la red o aumentar de manera significativa la capacidad de procesamiento del sistema.

La ventaja de este tipo de IDS es que puede detectar los ataques que se están

produciendo en tiempo real y responder a ellos antes de que ocasionen grandes daños.

Además, el proceso de auditoría posterior es, por lo general, menos costoso debido a que todo el tráfico ha sido ya analizado

El problema de estos sistemas es la poca precisión en la detección que suelen presentar debido al análisis superficial de los datos que realizan. De ahí que se haya que buscar un equilibrio entre la capacidad de la red, la capacidad de cómputo del sistema y la profundidad del análisis de los paquetes.

2.5.6.2. IDS basado en intervalos

Estos sistemas realizan el análisis de los paquetes a intervalos, es decir, sólo analizan el tráfico recogido en ciertos períodos de tiempo. Es un proceso parecido al análisis forense pero con una periodicidad fija.

Con estos sistemas el tráfico se analiza con algoritmos más precisos a la hora de detectar intrusiones. Son muy efectivos para detectar intrusiones recurrentes, identificar intrusiones exitosas y definir los privilegios de los usuarios. Además, el coste computacional es menor al tener una cantidad menor de datos para procesar.

En cuanto a la parte negativa, por lo general estos sistemas detectan los ataques una vez han ocurrido por lo que sólo sirven para evitarlos en el futuro. Además el primer ataque puede afectar al IDS y dejar desprotegido al sistema sin que éste lo note.

3. TRABAJOS RELACIONADOS

A continuación vamos a hacer un pequeño repaso a los trabajos más representativos que siguen esta línea de desarrollo: NIDS basados en análisis del Payload, NIDS basados en la búsqueda el gen de auto-replicamiento, NIDS basados en Random Forest y Sistemas Inmunológicos Artificiales.

3.1 NIDS basados en análisis del payload.

Estos sistemas detectan el código malicioso analizando el payload de los paquetes que circulan por la red. Constan típicamente de una fase de entrenamiento y otra de detección. La fase de entrenamiento se realiza con tráfico limpio que represente estadísticamente al tráfico usual del sistema. De esta forma, se crea un patrón de dicho tráfico.

Por otro lado, durante la fase de detección, el tráfico a analizar se modela y se compara con estos patrones para determinar si puede ser clasificado como peligroso.

En general, todos los sistemas englobados dentro de esta línea de investigación podrían considerarse variantes de PAYL [29], una de las primeras propuestas que usó esta técnica con éxito.

El sistema PAYL clasifica el tráfico basándose en 3 características: el puerto, el tamaño del paquete y la dirección de flujo (entrada o salida). Mediante estos 3 parámetros clasifican los payloads creando una serie de patrones para definir lo que sería un comportamiento normal dentro de cada clase. PAYL utiliza 1-gram para el análisis de estos patrones debido a la enorme cantidad de patrones que se crearán durante el entrenamiento del sistema. La razón de usar un tamaño de n-gram tan bajo es para evitar un tamaño de memoria demasiado grande. Aún así, los resultados con 1-gram son significativos.

Poseidon [7]. Este algoritmo se desarrolló para corregir los errores que surgen en la construcción de modelos en PAYL cuando se aplica clustering sobre el tamaño de los paquetes. El funcionamiento principal del sistema es similar al de PAYL, si bien añaden algunas mejoras, especialmente el uso de mapas auto organizativos (SOM) para clasificar el tráfico. Así, los paquetes son pre-procesados por medio del SOM que da como salida las diferentes clases en que deben clasificarse los paquetes. En este modelo, en lugar de clasificar los paquetes atendiendo al puerto, dirección y tamaño de los mismos, se clasifican teniendo en cuenta el puerto, la dirección y las clases devueltas por SOM. Con esto se consigue reducir considerablemente el número de clases mejorándose los resultados obtenidos por PAYL.

Combinación de múltiples clasificadores de una clase [26]. Esta técnica, también basada en PAYL, fue desarrollada para eliminar la vulnerabilidad del sistema original ante los ataques mimicry [24].

El sistema tiene varios detectores de una sola clase o detectores *Support Vector Machine* (SVM). La precisión del sistema aumenta a medida que aumente el número de detectores. Los detectores de una clase sólo distinguen dos posibilidades: pertenencia o no a la clase. El entrenamiento de cada detector sólo se hace con muestras de la clase que va a detectar. De ahí la necesidad de tener varios detectores para abarcar diferentes clases.

PCNAD [41]. Esta técnica surge para corregir el defecto de PAYL de no poder procesar los paquetes grandes en las redes rápidas con la suficiente velocidad. La característica principal de este sistema es que no analiza todo el payload de los paquetes si no que los divide y reduce para realizar los patrones y las comparaciones. De esta forma se consigue procesar todo el tráfico en redes rápidas.

Para la creación de los patrones y la posterior comparación del tráfico se usa el mismo sistema que en PAYL, es decir, se usa 1-gram y se guarda la frecuencia

de aparición de los distintos bytes de cada clase. Se realiza un patrón para cada servicio que tiene el terminal, por lo cual se debe entrenar el sistema en cada máquina en que se ejecute.

Para dividir el payload se usa el algoritmo de particionado CPP: el tráfico se va procesando con una ventana de 1-gram. Para cada 1-gram se ejecuta el rabin fingerprint, es decir, se obtiene un número entero que lo representa. A continuación se mira si este número cumple una condición asignada previamente. Por ejemplo, que el módulo 1000 del número esté entre 100 y 200.

Los 1-gram cuyos fingerprints cumplan dicha condición serán los que delimiten las particiones del payload que se analizarán. A la hora de construir los perfiles de cada servicio se define un parámetro n que determina el número de particiones del payload que se analizarán (se procesarán las n primeras particiones de cada paquete).

Con un valor adecuado de n, el perfil obtenido mediante el análisis de las particiones es muy similar al obtenido si se hubiera utilizado todo el payload. Durante la fase detección, los paquetes también son divididos mediante el algoritmo CPP y se usan sus n primeras particiones para realizar las comparaciones con los perfiles, teniendo n el mismo valor que en la fase de entrenamiento. Para realizar las comparaciones de los perfiles se usa la distancia reducida de Mahalanobis, al igual que en PAYL.

Anagram [44]. Este algoritmo es otra evolución de PAYL, desarrollada por los mismos autores para corregir las deficiencias que tenía el sistema original. Al igual que en PAYL, el sistema se basa en n-grams para procesar los paquetes y crear los patrones de comportamiento. Sin embargo, usa Bloom Filters para poder dividir los paquetes en n-grams de tamaños mayores que 1 sin que el coste en espacio y rendimiento del sistema se vea perjudicado.

El Bloom Filter es una estructura para almacenar de forma binaria la aparición o no de un determinado n-gram pero no el número de veces que aparece, es decir, en este caso no se almacena la frecuencia de distribución de cada byte sino su aparición dentro de los paquetes. Durante la fase de entrenamiento se rellenan dos Bloom Filters: uno con el tráfico normal y legítimo de la red, y otro con los n-grams que aparecen en virus conocidos. El proceso para rellenar esta última estructura debe ser supervisado debido a la utilización de ataques. Es por esto que se dice que el entrenamiento del sistema es semi-supervisado.

Durante la fase de detección se obtienen los n-grams de cada paquete y se comprueba si aparecen en alguno de los dos Bloom Filters. A continuación, se asigna una puntuación a este paquete teniendo en cuenta los n-grams que no aparecen en el patrón de tráfico legítimo y los que aparecen en el patrón del código malicioso. Si esta puntuación supera un determinado umbral, el paquete se considera un ataque.

El uso de n-grams más grandes permite una mayor precisión ya que se tienen en cuenta las dependencias entre bytes. Por otro lado, al no tener en cuenta la frecuencia de aparición de los n-grams pueden detectarse ataques ocultos entre tráfico legítimo. Aunque mejora los resultados de PAYL requiere de un entrenamiento muy controlado, puesto que no tolera ruido durante dicho entrenamiento.

3.2 Búsqueda del gen de auto-replicamiento

Una de las características que definen a la mayoría de programas maliciosos es que intentan autoreplicarse para expandirse por la red. Los métodos para que un programa pueda reproducirse son conocidos y relativamente pocos (alrededor de 50).

Esta técnica [38] [46] analiza el código de los programas buscando estos métodos (gen de autoreplicamiento) para detectar código malicioso.

La principal dificultad es que las instrucciones de autoreplicación pueden estar ocultas entre el resto de instrucciones o, aún peor, estar cifradas. Por tal motivo la búsqueda del "gen de autoreplicación" puede entenderse como la búsqueda de una serie de palabras dentro de un array de letras teniendo en cuenta los siguientes puntos:

- Todas las palabras se posicionan a lo largo de un *string* y deben leerse de izquierda a derecha, en orden de ejecución.

- Las palabras pueden estar fragmentadas. Por ejemplo, si se busca la palabra REPLICACION, ésta puede estar dividida en dos partes: RE y PLICACION. Esto implica que una vez encontrada la primera letra (R) hay que ser capaz de ver si con el resto de letras se puede completar la palabra. Para ello existen diversas técnicas de descifrado [7].

- El código malicioso puede llegar parcialmente codificado y decodificarse en un orden determinado. Por ello, durante el proceso de decodificación se tienen que hacer diversas interrupciones para analizar la composición de la imagen del ejecutable.

3.3 Random Forest

Sistema híbrido [13] que combina detección por patrones con detección por anomalías. Aunque también detecta código malicioso está pensado para detectar cualquier ataque o intrusión que pueda sufrir una red.

Este sistema se caracteriza por la utilización de random forest para la clasificación del tráfico durante las fases de entrenamiento y detección.

El algoritmo de clasificación del tráfico se basa en la creación de múltiples árboles de decisión. Estos árboles tienen características discriminantes en cada uno de sus nodos de forma que una muestra recorrerá el árbol desde la raíz hasta alguna de las hojas. En cada nodo intermedio la muestra se evalúa y elige el camino que mejor se ajuste a sus características. Las hojas del árbol son las diferentes clases en las que se dividirá el tráfico, es decir, que cuando una muestra llega a una hoja, ésta define la clase a la que pertenece.

La ventaja de usar árboles de decisión para la clasificación es la rapidez en que se procesa el tráfico. Sin embargo, para mantener esta rapidez y no saturar el sistema los árboles no deben ser muy grandes ni complejos. En caso contrario, se obtendría poca precisión a la hora de clasificar. Para solucionar este problema se usa el algoritmo de random forest.

El algoritmo de random forest consiste en crear varios árboles de decisión distintos pero similares en cuanto a las características usadas en sus nodos. Para clasificar, una muestra se introduce en cada uno de estos árboles y una vez obtenido los resultados se calcula la moda, es decir, se calcula la hoja más repetida que ha alcanzado la muestra en los árboles. Esta hoja será la clase a la que pertenezca la muestra.

El algoritmo se caracteriza por dos parámetros fundamentales: el número de árboles que compondrán el bosque y las características que se usarán en los nodos para discriminar las muestras. Además, para aumentar el rendimiento y la precisión del sistema se usan otras técnicas como:

- El *mecanismo outlier*, que se encarga de medir el grado de pertenencia de una muestra a una clase determinada. Se basa en el cálculo de un tipo de distancia entre la muestra y todas las clases. Se usa en el módulo de detección de anomalías.

- El *algoritmo de selección de características* se encarga de determinar cuáles serán las características que usen los árboles para clasificar los paquetes. Para ello analiza todas las características disponibles, las evalúa otorgándoles una puntuación y elige las más apropiadas, es decir las de mejor puntuación. Normalmente se define un umbral para determinar si una característica es buena.

- Las *técnicas de sampling* se encargan de equilibrar las muestras de ataques usadas en el aprendizaje para obtener una mayor precisión en la detección posterior. Se intenta que la frecuencia de aparición sea parecida para entrenar al sistema contra los ataques poco comunes que suelen ser los más peligrosos.

3.4 Sistemas inmunológicos artificiales.

Una de las técnicas investigadas en los últimos años consiste en imitar el sistema biológico inmunitario (SBI), es decir, en tratar una intrusión en un sistema informático de la misma forma que los seres vivos tratan sus infecciones.

El SBI es capaz de detectar organismos extraños y eliminarlos sintetizando unas proteínas llamadas anticuerpos que tienen, a su vez, varios tipos de detectores. Cuando un anticuerpo detecta un intruso se multiplica para ser más efectivo hasta que la amenaza es eliminada. Después de esto la mayoría de los anticuerpos mueren, aunque algunos permanecen creando una memoria inmunitaria para poder reaccionar rápidamente en caso de contraer de nuevo la infección.

Las técnicas que se engloban dentro de este tipo de sistemas [6][21][28][32][33][36] intentan imitar este comportamiento para detectar el código malicioso.

3.5 Otras propuestas

Rate Limiting [43] [45] detecta comportamiento anómalo en las conexiones basándose en la premisa de que un host infectado tratará de conectarse a muchas máquinas diferentes en un período corto de tiempo. Esta propuesta detecta PortScans colocando a aquellas conexiones que exceden cierto umbral en una cola.

El Algoritmo *Threshold Random Walk* (TRW) [15] se basa en que la probabilidad de que un intento de conexión sea exitoso es mayor para un programa benigno que para uno anómalo. Utiliza pruebas de hipótesis secuenciales para clasificar si un host remoto es un scanner o no.

TRW con Ratio Límite (TRW-CB) [30] es una solución híbrida que busca complementar las dos soluciones anteriores. Es un detector que limita la tasa en la que nuevas conexiones son inicializadas aplicando pruebas de hipótesis secuenciales en orden inverso al cronológico. El algoritmo frenará aquellos hosts que han tenido conexiones fallidas.

El *Método de Máxima Entropía* [12] estima la distribución del tráfico benigno usando una estimación de máxima entropía. En el período de entrenamiento se clasifica a los paquetes en clases distintas y se determina una distribución para cada clase. Luego se observa la distribución de los paquetes en tiempo real y se compara con la distribución base. Se dispara una alarma cuando una clase supera un umbral de diferencia.

Detección de anomalías en la cabecera de los paquetes (PHAD) [19] [20] utiliza algunos campos del encabezado tales como direcciones IP, puertos, protocolos y banderas TCP. Durante la etapa de entrenamiento se le asigna una puntuación a cada valor de cada uno de los campos del encabezado y luego se suman para obtener una puntuación total de anomalía para cada paquete. Los primeros n paquetes con mayor puntuación son considerados peligrosos.

El *Sistema Experto de Detección de Intrusiones de Nueva Generación* (NIDES)

[2][3] es un detector estadístico que compara los perfiles de tráfico recolectados durante un gran período de tiempo con perfiles recolectados en un corto período de tiempo sobre datos en tiempo real. Dispara una alarma si existe una desviación considerable entre ambas distribuciones.

4. SISTEMA INTELIGENTE DE DETECCIÓN DE INTRUSIONES

El Sistema Inteligente de Detección de Intrusiones es un sistema de detección de intrusiones en busca de Malware basado en el análisis del payload del tráfico de la red. Su objetivo es detectar el envío de malware antes de que se produzca la instalación del mismo en la máquina de la víctima.

Este sistema realiza un análisis de los payloads de los paquetes que circulan por la red en busca de anomalías estadísticas con respecto al tráfico usual de dicha red, considerando dichas anomalías como paquetes correspondientes a algún tipo de Malware.

El sistema consta de dos fases: entrenamiento y detección. Durante la fase de entrenamiento se construye un patrón del tráfico que circula por una determinada red y, posteriormente se compara dicho modelo con una batería de ataques para determinar qué distribuciones corresponden a tráfico legítimo y qué distribuciones corresponden a ataques, es decir, se obtiene un conjunto de reglas que son capaces de clasificar el tráfico de la red.

Se podría pensar que cuanto mayor sean el tráfico limpio disponible y el número de muestras de la batería de ataques para el entrenamiento, mejores serán los resultados. Sin embargo, el Sistema desarrollado ha demostrado que no es necesaria una cantidad demasiado grande de tráfico durante el entrenamiento para obtener excelentes resultados en los análisis y que las mejoras que se producen incrementando en grandes cantidades los datos del entrenamiento no influyen significativamente en los resultados de los análisis posteriores.

El diseño ideal de un sistema de detección de intrusos basado en el análisis de los payloads consistiría en almacenar el contenido de todos los payloads de los paquetes limpios que circulan usualmente por una red y posteriormente comparar el tráfico a analizar de la red en cuestión con dichos payloads

almacenados. De esta manera, todo payload que no aparezca durante el entrenamiento pasa a considerarse Malware.

Parece lógico suponer que un sistema de estas características entrenado de una forma adecuada (hay que reconocer que el entrenamiento debería ser lo más exhaustivo posible) obtendría unos resultados tanto en detección de Malware como en falsos positivos muy satisfactorios (presumiblemente muy cercanos al 100% de detección ante ataques con un valor muy cercano al 0% de falsos positivos).

Sin embargo, este sistema a día de hoy es irrealizable, puesto que si consideramos payloads de 1500 bytes de tamaño tendríamos $2.29 * 10^{3612}$ (2^{1500*8}) posibles combinaciones. Una estructura que fuese capaz de almacenar todas estas distintas posibilidades resulta inviable actualmente debido a limitaciones tecnológicas.

Por lo tanto, para realizar el modelado del tráfico de la red se ha optado por reducir el número de combinaciones utilizando la técnica de los n-grams así como una variación de la técnica conocida como Bloom Filters, obteniendo un diseño viable con la tecnología actual con unos resultados satisfactorios.

A continuación describiremos las técnicas de N-Gram y Bloomfilter, así como el tamaño elegido para los mismos.

Posteriormente detallamos el funcionamiento del sistema que se basa en Redes Neuronales SOM (Self-Organizing Maps) y Redes Neuronales LVQ (Learnig Vector Quantization) para la creación de reglas.

4.1. Uso de N-Gram

La técnica n-gram [9] [18] es una técnica utilizada en el procesamiento del lenguaje natural. Su principal objetivo es la predicción, dentro de una sucesión de elementos, del siguiente elemento conociendo los anteriores y las distintas

probabilidades o distribuciones de aparición. Una de sus múltiples variantes consiste en utilizar esta técnica para resumir una cierta cantidad de datos que de otra manera sería imposible analizar o almacenar, ya sea por las limitaciones tecnológicas respecto al almacenamiento de datos, velocidad de análisis mínima requerida por parte del sistema o capacidad máxima de procesamiento.

La manera de aplicar la técnica de n-gram en el sistema diseñado es la siguiente: Se elige un cierto tamaño n (en este caso 3) que determinará el número de bytes de cada n-gram. El sistema actuará como una ventana de tamaño 3 por la que irán pasando los bytes del paquete de forma secuencial (los bytes del payload del paquete). Cada 3-gram obtenido de un paquete, es decir, cada secuencia de 3 bytes consecutivos que tenga el paquete, se procesará como si fuese una entidad propia. El resultado de procesar un paquete vendrá determinado por la unión de los resultados obtenidos de analizar cada uno de los n-grams en los que se pueda dividir dicho paquete.

Esta técnica permite resumir el contenido de los paquetes y buscar dependencias y similitudes entre ellos. La principal ventaja de su uso es el poco consumo de recursos y facilidad para implementarla. Por otro lado, el inconveniente es la generación de muchos n-grams lo que requiere alguna técnica auxiliar que permita resumir la información.

4.2. Uso de BloomFilter

BloomFilter [1][8][10][11][17][23][27] es una estructura de datos probabilística utilizada para determinar si un elemento pertenece a un conjunto de datos o no.

En el sistema diseñado se utiliza de la siguiente manera: se considera el Bloom Filter como una estructura de datos cuyo objetivo es almacenar la información sobre los n-grams que aparecen en un paquete. Es posible imaginar el Bloom Filter como un array en el que cada posición representa un tipo de n-gram distinto.

En la práctica, para implementar esta idea se consideró el Bloom Filter como un array de bits en el que un 0 representaba que ese n-gram no aparecía en el tráfico normal de la red y un 1 representaba que ese n-gram si había aparecido. Al usar bits para representar n-grams se consigue reducir en gran medida la memoria necesaria para guardar la información sobre el tráfico legítimo.

Sin embargo, esta implementación tuvo que ser desechada por el hecho de que debido a la especial composición de n-grams de los payloads del tráfico, al realizar el entrenamiento se rellenaba a 1 prácticamente por completo la estructura del Bloom Filter, por lo que, en la práctica, casi la totalidad de los n-grams eran considerados como legítimos, de modo que el número de ataques detectados era muy reducido.

Para solucionar este problema se consideró que se pusieran a 0 aquellas posiciones del Bloom Filter que en menor medida intentasen ponerse a 1 durante el entrenamiento. Para determinar la medida que indicaba si un determinado n-gram había aparecido el número suficiente de veces como para ponerlo a 1 se utilizó la desviación típica.

Esta modificación del concepto del Bloom Filter dio buenos resultados en la práctica. Sin embargo, se planteó la siguiente reflexión: si para calcular la desviación típica se necesita almacenar temporalmente en una estructura cuántas veces se quería poner a 1 cada posición del Bloom Filter, no tenía sentido resumir posteriormente dichos datos entre dos posibilidades: había aparecido durante el entrenamiento un número suficiente de veces (1) o no había aparecido un número de veces suficiente durante el entrenamiento (0).

En su lugar, se decidió almacenar directamente cuantas veces se intentaba poner a 1 una posición del Bloom Filter durante el entrenamiento y tener en cuenta dicha distribución para determinar a partir de qué umbral se considera que estamos analizando el payload de un ataque o no.

4.3. Consideraciones sobre el Tamaño de los N-Gram

Para realizar el análisis de los paquetes, debido a limitaciones de los equipos actuales, se dividen en pequeñas porciones llamadas n-grams. El tamaño seleccionado de n-gram es de 3 bytes. La elección de este tamaño de n-gram no es aleatoria.

A continuación se explican los motivos de dicha elección y, cómo influye a su vez, el tamaño elegido para los n-grams en el tamaño de la estructura denominada BloomFilter.

Para almacenar el número de apariciones de cada n-gram durante el entrenamiento se utiliza una estructura (BloomFilter), de manera que a cada n-gram distinto que se pueda encontrar durante el entrenamiento se le asocia una posición diferente del BloomFilter. Por ello se utiliza una correspondencia directa entre cada n-gram y el BloomFilter, de manera que el valor del n-gram determina su posición dentro del BloomFilter.

Como es lógico pensar, cuanto mayor sea el tamaño del BloomFilter mejores serán los resultados obtenidos, puesto que el sistema será capaz de diferenciar entre un mayor número de n-grams. Sin embargo, como existe una correspondencia directa entre los n-grams y el BloomFilter, no se puede utilizar un n-gram excesivamente grande, puesto que el tamaño del BloomFilter está relacionado con el tamaño del n-gram (véase tabla 1), y tendríamos un Bloomfilter cuyo tamaño rabasaría las posibilidades que ofrece la tecnología actual.

n-gram (bytes)	BloomFilterX
1-ngram	256 (2^8) posiciones
2-ngram	65536 (2^{16}) posiciones
3-ngram	16777216 (2^{24}) posiciones
4-ngram	4294967296 (2^{32}) posiciones

…	…

Tabla 1. Relación entre el tamaño del n-gram y del BloomFilterX

Para determinar el tamaño adecuado del n-gram en nuestro sistema tuvimos en cuenta dos aspectos:

El BloomFilter es un array de tipo entero. Su tamaño se indica en la tabla 2 según la opción elegida.

n-gram (bytes)	**Tamaño en Bytes**
1-ngram	1024 B
2-ngram	262144 B
3-ngram	67108864 B (64 MB)
4-ngram	17179869184 B (16 GB)
…	…

Tabla 2. Tamaño del BloomFilterX para cada n-gram elegido

En la tabla 2 se observa que, dependiendo del sistema la duda podría estar entre la elección de un 3-gram o un 4-gram, puesto que un mayor n-gram sería inaceptable con los sistemas actuales.

Sin embargo, el lenguaje de programación (C en nuestro caso) imposibilita la creación de arrays de 232 posiciones (el necesario para 4-gram).

Por todo lo anterior se eligió un tamaño de n-gram de 3 bytes.

4.4. Especificación del Sistema

4.4.1. Fases del Sistema

El Sistema Inteligente de Detección de Intrusiones consta de dos modos de ejecución o fases: Fase de Entrenamiento (dividida a su vez en cuatro etapas que detallaremos a continuación) y Fase de Detección.

4.4.2. Fase de Entrenamiento

En la fase de entrenamiento se crea un modelo estadístico del tráfico legítimo de la red utilizando las técnicas de n-gram, BloomFilter y Redes Neuronales SOM (Self-Organizing Maps) y Redes Neuronales LVQ (Learnig Vector Quantization):

Esta fase a su vez se divide en 4 etapas: Entrenamiento-Bloomfilter Entrenamiento-SOM, Etiquetado de los cluster's y Entrenamiento-LVQ.

- **Entrenamiento Bloomfilter.** En esta etapa el sistema de pone a cero creando las estructuras necesarias y se rellena el Bloomfilter del sistema.

- **Entrenamiento-SOM.** La finalidad de esta etapa es realizar una clusterización de los datos de entrada.

- **Etiquetado de los Cluster's**. Se realiza un etiquetado automático de los cluster's creados por la red neuronal SOM en función de la proporción de paquetes de ataque y de tráfico limpio que contenga cada cluster o clase.

- **Entrenamiento-LVQ**. Esta etapa del entrenamiento se utiliza para afinar el etiquetado realizado en la etapa anterior de los cluster's creados por la red SOM.

4.4.2.1. Entrenamiento Bloomfilter

En esta etapa se inicializa la estructura principal necesaria para realizar el modelado del tráfico de la red.

BloomFilter: Almacena el número de apariciones de cada n-gram (en la etapa conocida como "Entrenamiento-Base"). El sistema se basa en la idea de que el tráfico legítimo contendrá n-grams que han aparecido durante el entrenamiento en mayor medida que los n-grams que contendrá el tráfico malicioso. A su vez, los experimentos realizados arrojaron otra característica destacable que también es utilizada para la detección de malware: los payload correspondientes a malware tienen una distribución de n-grams menos homogénea que los payloads correspondientes a tráfico legítimo.

Posteriormente se procede a rellenar dicha estructura utilizando dataset's de tráfico limpio.

Cabe destacar, que en el Sistema Inteligente de Detección de Intrusiones, para determinar cuándo podemos dar por concluida esta fase del entrenamiento hemos utilizado el Coeficiente de Correlación de Pearson, puesto que al ser una fórmula estadística universal la interpretación por parte de los usuarios del Sistema de los resultados que arroje será más sencilla.

No cabe duda de que el Bloomfilter es uno de los puntos clave de nuestro sistema puesto que representa un "resumen" estadístico del tráfico que presenta la nuestra red y en caso de que dicho resumen no sea preciso el sistema de detección de ataques no podrá ser de ninguna manera eficiente.

Por lo tanto, tenemos que ser capaces de demostrar dos características que tiene que cumplir nuestro sistema:

Debemos de estar seguros que puede realizarse dicha representación o resumen del tráfico.

Se tiene que poder determinar cuándo el Bloomfilter alcanza en la fase de entrenamiento un estado de "equilibrio" a partir del cual no sea necesario continuar con el entrenamiento.

Para demostrar que se puede construir una representación del tráfico de la red mediante un Bloomfilter y que se puede puede determinar cuando hemos terminado de entrenar el sistema, de manera que aún continuando con el entrenamiento no se viese afectada su distribución interna llegando al punto óptimo de entrenamiento, vamos a realizar de serie de ensayos o experimentos.

Para realizar estas pruebas contamos con 12 datasets de tráfico limpio con un tamaño que oscila entre los 250 MB y los 350MB. A partir de estos datasets hemos ido construyendo distintos conjuntos de entrenamiento compuestos en primer lugar por un único dataset, a continuación con dos, posteriormente con tres... y así sucesivamente tal y como reflejamos en la tabla 3.

BloomFilter	exp01	exp02	exp03	exp04	exp05	exp06
01x	1	6	11	4	9	2
02x	2	7	12	5	10	3
03x	3	8	1	6	11	4
04x	4	9	2	7	12	5
05x	5	10	3	8	1	6
06x	6	11	4	9	2	7
07x	7	12	5	10	3	8
08x	8	1	6	11	4	9
09x	9	2	7	12	5	10
10x	10	3	8	1	6	11
11x	11	4	9	2	7	12
12x	12	5	10	3	8	1

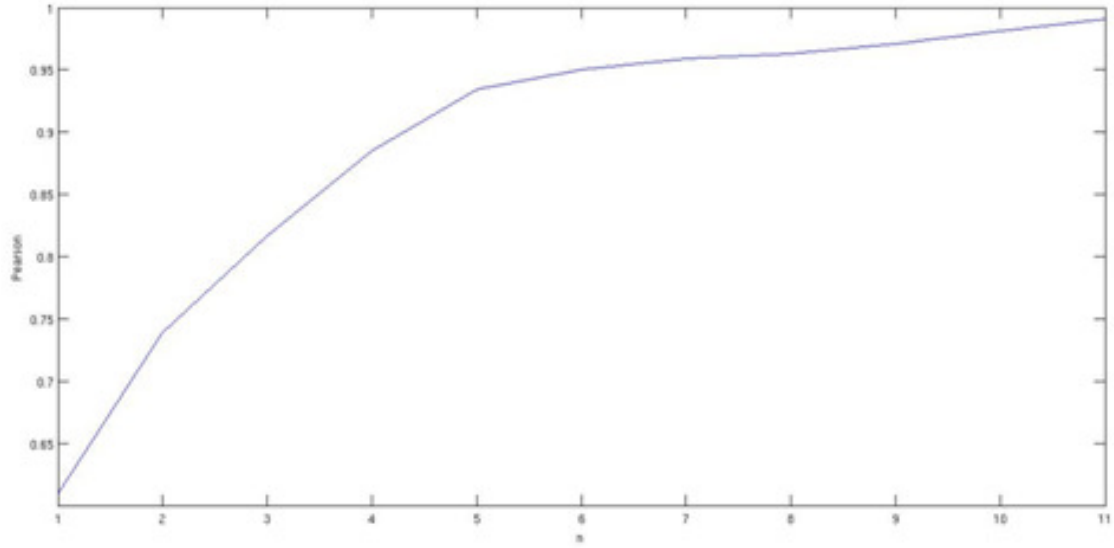
Tabla 3. 12 datasets de tráfico limpio

Para interpretar esta tabla, tenemos que aclarar que los Bloomfilter del experimento 2 se crean a partir de los del experimento 1, los Bloomfilter del experimento 3 a partir de los del experimento 2 y así sucesivamente. De esta manera, por ejemplo, en el experimento exp02, el Bloomfilter0102 será el resultado de utilizar como entrenamiento los dataset's 1 y 6. En el experimento

exp4 el Bloomfilter0104 será el resultado de utilizar como entrenamiento los dataset's 1, 6, 11 y 4.

Por otro lado, para comparar la distribución de los distintos dataset's de cada experimento que hemos construido a lo largo de las pruebas se ha utilizado el Coeficiente de Correlación de Pearson (fórmula ya implementada en Matlab o en la mayoría de hojas de cálculo), de manera que cuanto más se aproxima este coeficiente a 1 más similares podemos considerar los dos Bloomfilters que se están comparando. Aunque el punto ideal de parada del entrenamiento sería cuando obtuviésemos un 1 en el Coeficiente de Correlación de Pearson al comparar dos Bloomfilters construidos con distintos datasets, en la práctica obtener un valor alto cercano a 1 puede considerarse un resultado muy satisfactorio, y razón suficiente para dar por concluido el entrenamiento.

En la figura 5 se muestra cómo evoluciona el Coeficiente de Correlación de Pearson en función del número de dataset's a partir de los cuales construyamos los Bloomfilters.



	exp01	exp02	exp03	exp04	exp05	exp06	exp07	exp08	exp09	exp10	exp11
Pearson	**0.6090**	**0.7391**	**0.8165**	**0.8850**	**0.9341**	**0.9501**	**0.9589**	**0.9630**	**0.9707**	**0.9815**	**0.9912**

Fig. 5. Segunda alternativa de determinación de las Ks

Como podemos ver en la figura 5, el coeficiente de correlación de Pearson comienza a estabilizarse en torno al valor de 0.95, que se alcanza en el exp06, entrenando el Bloomfilter con seis dataset's. Sin embargo, puede apreciarse también una aceleración posterior a partir del uso de nueve dataset's para entrenar el Bloomfilter. Este hecho se debe a las características propias del método de construcción de Bloomfilter's que estamos empleando, puesto que al utilizar cada vez más dataset's para entrenar el Bloomfilter más se repiten los dataset's para cada nivel de entrenamiento, de manera que al utilizar once dataset's, únicamente se diferencia cada Bloomfiter en un dataset, mientras que los diez restantes utilizados para construir el Bloomfilter son similares. Esta característica puede verse mejor en la tabla 4, para exp11, que indica con una cruz si el dataset se ha utilizado en la construcción del Bloomfilter__11.

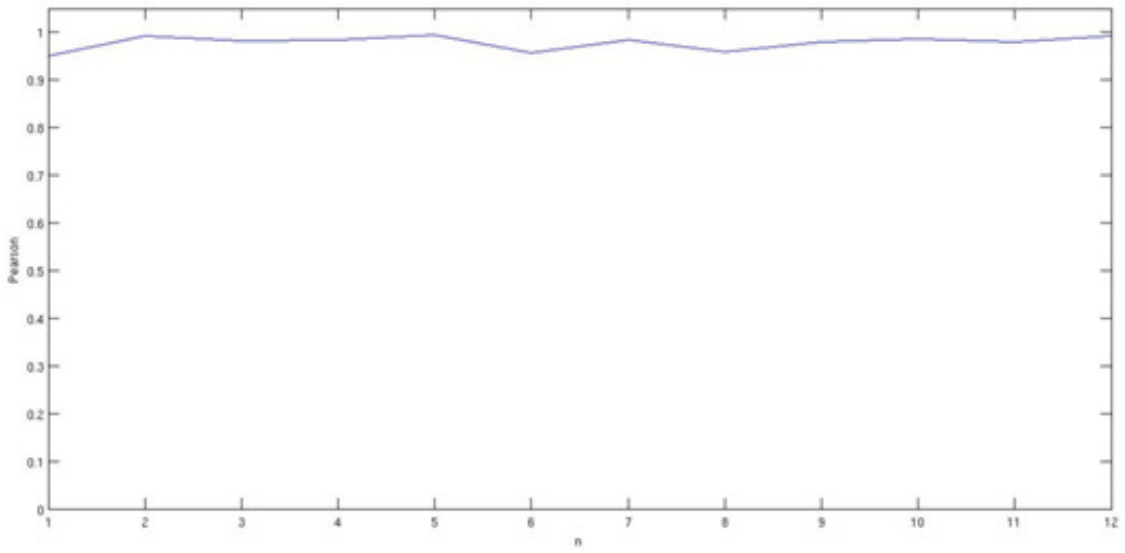
BloomFilter	**datasets**											
	01	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**10**	**11**	**12**
0111	X	X	X	X	X	X	X		X	X	X	X
0211	X	X	X	X	X	X	X	X		X	X	X
0311	X	X	X	X	X	X	X	X	X		X	X
0411	X	X	X	X	X	X	X	X	X	X		X
0511	X	X	X	X	X	X	X	X	X	X	X	
0611		X	X	X	X	X	X	X	X	X	X	X
0711	X		X	X	X	X	X	X	X	X	X	X
0811	X	X		X	X	X	X	X	X	X	X	X
0911	X	X	X		X	X	X	X	X	X	X	X
1011	X	X	X	X		X	X	X	X	X	X	X
1111	X	X	X	X	X		X	X	X	X	X	X
1211	X	X	X	X	X	X		X	X	X	X	X

Tabla 4. Datasets utilizados en la construcción del Bloomfilter__11

Por ello, a raíz de las pruebas realizadas, podemos afirmar que a partir del uso de seis dataset's para el entrenamiento del Bloomfilter la composición de los mismos se estabiliza, es decir, que se necesita aproximadamente 2 GB de tráfico para entrenar el sistema.

Para demostrar esta suposición se han comparado los Bloomfilter´s construidos a partir de seis dataset´s con los Bloomfilter´s construidos con once dataset´s. De esta manera se comparaba la distribución del Bloomfilter0106 con el Bloomfilter0111, el Bloomfilte0206 con el Bloomfilter0211, etc. y así sucesivamente. En la figura 6 podemos observar los resultados de este experimento.

Si calculamos la media de los puntos de la figura 6 obtenemos un valor para el Coeficiente de Correlación de Pearson de 0.9787. A partir de los resultados de este experimento podemos concluir que con exp06 se alcanza el punto óptimo de entrenamiento, puesto que los resultados obtenidos al comparar los Bloomfilter's del experimento seis con los del experimento once los resultados son similares a los que se obtienen al comparar los Bloomfilters del nivel nueve entre ellos mismos, 0.9707.



exp06	Bloomfilter

vs exp11	0106 Vs 0111	0206 Vs 0211	0106 Vs 0111	0206 Vs 0211	0106 Vs 0111	0206 Vs 0211	0106 Vs 0111	0206 Vs 0211	0106 Vs 0111	0206 Vs 0211	0106 Vs 0111	0206 Vs 0211
Pearson	**0.9501**	**0.9923**	**0.9817**	**0.9852**	**0.9939**	**0.9565**	**0.9852**	**0.9594**	**0.9870**	**0.9871**	**0.9793**	**0.9929**

Fig. 6. Bloomfilter's construidos de 6 dataset's Vs Bloomfilter´s construidos con 11 dataset's

Para demostrar esta afirmación vamos a analizar los resultados que se obtuvieron al comparar los Bloomfilters del exp06 entre ellos mismos y los Bloomfilters del exp09 entre ellos mismos, reflejando también en cada caso cuantos datasets comparten entre los dos Bloomfilters que estamos comparando.

Primero comenzaremos con el experimento número 6, exp06.

En la tabla 5 podemos ver que lógicamente cuantos más dataset's comparten los Bloomfilters que estamos comparando mayor es el Coeficiente de Correlación de Pearson que se obtiene de la comparación. Sin embargo, la diferencias son más pequeñas de lo que podríamos pensar. En la tabla 5 se muestra la media de los casos clasificados según el número de dataset's que comparten en su entrenamiento, los que no comparten ningún dataset, los que comparten un dataset, los que comparten dos dataset's... y así sucesivamente hasta los que comparten cinco datasets.

EXP06	1	2	3	4	5	6	7	8	9	10	11	12
1	1	1 – 0,9088	4 – 0,9899	3 – 0,9217	2 – 0,9713	5 – 0,9655	0 – 0,8472	5 – 0,9775	2 – 0,8785	3 – 0,9807	4 – 0,9024	1 – 0,9181
2	0	1	1 – 0,9339	4 – 0,9756	3 – 0,9651	2 – 0,9299	5 – 0,9837	0 – 0,9163	5 – 0,9885	2 – 0,9475	3 – 0,9822	4 – 0,9945
3	0	0	1	1 – 0,9343	4 – 0,9882	3 – 0,9666	2 – 0,8843	5 – 0,9794	0 – 0,9007	5 – 0,9932	2 – 0,9198	3 – 0,9436
4	0	0	0	1	1 – 0,9479	4 – 0,9679	3 – 0,9673	2 – 0,9482	5 – 0,9815	0 – 0,9498	5 – 0,9916	2 – 0,9673
5	0	0	0	0	1	1 – 0,9535	4 – 0,9260	3 – 0,9659	2 – 0,9374	5 – 0,9861	0 – 0,9390	5 – 0,9747
6	0	0	0	0	0	1	1 – 0,9024	4 – 0,9832	3 – 0,9253	2 – 0,9702	5 – 0,9517	0 – 0,9235
7	0	0	0	0	0	0	1	1 – 0,8885	4 – 0,9937	3 – 0,9049	2 – 0,9822	5 – 0,9784
8	0	0	0	0	0	0	0	1	1 – 0,9107	4 – 0,9763	3 – 0,9345	2 – 0,9285
9	0	0	0	0	0	0	0	0	1	1 – 0,9193	4 – 0,9924	3 – 0,9828
10	0	0	0	0	0	0	0	0	0	1	1 – 0,9366	4 – 0,9535
11	0	0	0	0	0	0	0	0	0	0	1	1 – 0,9752
12	0	0	0	0	0	0	0	0	0	0	0	1

Tabla 5. Experimento número 6

	exp01	exp02	exp03	exp04	exp05	exp06	exp07	exp08	exp09	exp10	exp11
Pearson	0.6090	0.7391	0.8165	0.8850	0.9341	0.9501	0.9589	0.9630	0.9707	0.9815	0.9912

Tabla 6. Coeficiente de Correlación de Pearson

En la tabla 6 podemos ver que aunque el Coeficiente de Correlación de Pearson aumenta, no es un aumento demasiado significativo.

Si observamos los resultados obtenidos en el experimento número nueve, exp09, en la tabla 7, podemos observar que los valores obtenidos no parecen a simple vista significativamente mayores que en el exp06.

Para profundizar en los datos obtenidos, en la tabla 8 podemos observar los Coeficientes de Correlación de Pearson para los casos en los que los Bloomfilters comparten seis datasets, siete datasets y ocho datasets.

Si nos fijamos detenidamente en las medias de coeficiente de correlación de Pearson obtenidos en los experimentos seis y nueve, y consideramos que el caso de exp06 en el que se comparten cuatro datasets en la construcción del Bloomfilter (se comparten cuatro datasets de seis que se utilizan, 4/6 => 66%) es similar al caso de exp09 en el que se comparten seis datasets en la construcción del Bloomfilter (se comparten seis datasets de nueve que se utilizan, 6/9 => 66%) podemos ver que los valores obtenidos para ambos casos son prácticamente idénticos, para exp06 0,9703 y 0,9666 para exp09.

EXP09	1	2	3	4	5	6	7	8	9	10	11	12
1	**1**	**6 – 0,9896**	**6 – 0,9563**	**6 – 0,9860**	**6 – 0,9751**	**8 – 0,9704**	**6 – 0,9829**	**8 – 0,9986**	**6 – 0,9875**	**6 – 0,9633**	**7 – 0,9815**	**6 – 0,9813**
2	**0**	**1**	**6 – 0,9552**	**6 – 0,9912**	**6 – 0,9776**	**6 – 0,9636**	**8 – 0,9884**	**6 – 0,9907**	**8 – 0,9982**	**6 – 0,9618**	**6 – 0,9869**	**7 – 0,9867**
3	**0**	**0**	**1**	**6 – 0,9369**	**6 – 0,9682**	**6 – 0,8751**	**6 – 0,9145**	**8 – 0,9553**	**6 – 0,9635**	**8 – 0,9981**	**6 – 0,9219**	**6 – 0,9153**
4	**0**	**0**	**0**	**1**	**6 – 0,9795**	**6 – 0,9829**	**6 – 0,9930**	**6 – 0,9880**	**8 – 0,9905**	**6 – 0,9438**	**8 – 0,9973**	**6 – 0,9930**
5	**0**	**0**	**0**	**0**	**1**	**6 – 0,9472**	**6 – 0,9692**	**6 –0,9755**	**6 – 0,9836**	**8 – 0,9730**	**6 – 0,9751**	**8 – 0,9744**
6	**0**	**0**	**0**	**0**	**0**	**1**	**6 – 0,9888**	**6 – 0,9719**	**6 – 0,9570**	**6 – 0,8871**	**8 – 0,9883**	**6 – 0,9880**
7	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**6 – 0,9850**	**6 – 0,9840**	**6 – 0,9243**	**6 – 0,9956**	**8 – 0,9982**
8	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**6 – 0,9890**	**6 – 0,9624**	**6 – 0,9831**	**6 – 0,9830**
9	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**6 – 0,9685**	**6 – 0,9853**	**6 – 0,9836**
10	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**6 – 0,9298**	**6 – 0,9262**
11	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**6 – 0,9962**
12	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**

Tabla 7. Experimento número 6

EXP09	6	7	8
Pearson	0.9666	0.9841	0.9858

Tabla 8. Coeficiente de Correlación de Pearson

Por tanto, se puede afirmar que, como sospechábamos, a partir de exp06 los valores obtenidos en el Coeficiente de Correlación de Pearson aumentan por las características propias del experimento (se comparten más datasets para la construcción de Bloomfilters que posteriormente se van a comparar) no porque aumentemos la cantidad de dataset's para el entrenamiento, de manera que llegados a este punto podemos afirmar que no es necesario continuar con esta etapa del entrenamiento.

4.4.2.2. Entrenamiento-SOM

La finalidad de la Red Neuronal SOM (Self-Organizing Map) utilizada es realizar una clusterización de los datos de entrada, de manera que el sistema sea capaz de clasificar cada uno de los paquetes del tráfico de la red que recibe como entrada.

El objetivo de este tipo de redes neuronales es dado una serie de vectores (patrones) para los que se supone una posible clasificación por ciertas características (reflejada en la distribución de probabilidad de los indivíduos):

Crear patrones para cada una de las clases presentadas, entendiendo como clase una colección de entradas con características similares.

Para los casos más frecuentes, crear un mayor número de clases, de manera que los reflejen con mayor capacidad de detalle.

Asociar las clases creadas de manera que se sitúen ordenadamente en la capa de salida según su relación entre ellas (crear mapas topológicos).

Una de las principales razones para utilizar este tipo de redes neuronales para realizar la clusterización de los datos es la cualidad que presentan de crear una representación espacial organizada de los patrones de entrada, de manera que son especialmente eficaces para reconocer patrones aún en situaciones con mucho "ruido".

En nuestro caso consideraremos como "ruido" aquello paquetes que son utilizados en el entrenamiento del sistema pero que han sido etiquetados de manera errónea. Por ejemplo, para entrenar nuestro sistema utilizamos una gran cantidad de paquetes limpios para rellenar el Bloomfilter.

Sin embargo, es lógico pensar que aunque la práctica totalidad de los paquetes utilizados efectivamente corresponderán a tráfico limpio, es posible que algunos paquetes, sin embargo, pertenezcan a algún tipo de malware. Igualmente es posible que aunque un paquete sea legítimo, se asemeje más a la mayoría de los paquetes correspondientes a ataques que a los paquetes de tráfico limpio.

De manera análoga, aunque la mayoría de los paquetes que componen el envío de malware estarán correctamente etiquetados como maliciosos, un número no despreciable de paquetes que son parte del envío de código malicioso serán etiquetados también como maliciosos cuando en realidad se asemejan más a los paquetes etiquetados como legítimos.

Estas características que presentan el tráfico de la red elevan la complejidad del sistema a utilizar para clusterizar el tráfico y poder clasificar los paquetes como maliciosos o legítimos. Por ello, poder utilizar una técnica de clusterización que tolere una cierta cantidad de "ruido" en el entrenamiento es crucial para el diseño de nuestro sistema, y consideramos el uso de redes neuronales SOM como la técnica más adecuada.

Otra característica secundaria que también presenta esta técnica de clusterización es que aunque el entrenamiento sea costoso computacionalmente (en nuestro caso esta fase de entrenamiento dura aproximadamente cinco horas) una vez que el sistema es entrenado, su capacidad para analizar datos es muy elevada, característica que permite a nuestro sistema proteger redes de comunicaciones con un flujo de datos elevado.

Esta característica es merecedora de tenerse en cuenta, puesto que aunque consiguiéramos diseñar y entrenar un sistema con unos resultados excelentes en cuanto a alta capacidad de detección de malware como mínima cantidad de falsos positivos generados, de nada serviría si su capacidad de análisis del tráfico de una red de telecomunicaciones no fuese lo suficientemente elevada.

A continuación vamos a describir las principales características de nuestra red neuronal SOM así como del entrenamiento utilizado para la misma:

- Red bidimensional de 196 neuronas (14*14).
- Nº de entradas: 8 "características".
- Topología hexagonal.
- Nº de ciclos de entrenamiento: 40.000.
- Nº de ataques utilizados en el entrenamiento: 44.

De las características anteriormente detalladas hay dos que pueden generar dudas en el lector llamar especialmente la atención: ¿por qué utilizar ataques en lugar de tráfico limpio para esta fase del entrenamiento? y ¿en qué consisten exáctamente las 8 "características" que utilizamos como entradas de la red neuronal?.

Como respuesta a la primera pregunta, utilizamos ataques en lugar de tráfico limpio puesto que se ha observado que de esta manera se consiguen mapas topológicos mucho más heterogéneos que el obtenidos utilizando tráfico limpio. Ello se debe a que el tráfico con ataques también contiene una gran cantidad de paquetes que pueden considerarse "limpios" mientras que el tráfico limpio contiene una proporción mucho menor de paquetes considerados como maliciosos.

En la práctica este hecho supone que para entrenar la red SOM si decidiésemos utilizar tráfico limpio necesitaríamos una cantidad de tráfico mucho mayor que utilizando tráfico con ataques de manera que el tiempo de entrenamiento se incrementaría de manera proporcional.

A continuación daremos respuesta a la segunda pregunta: ¿en qué consisten exactamente las 8 "características" que utilizamos como entradas de la red neuronal?. Una vez que se ha construido el BloomFilter, utilizamos tráfico de ataques , pero esta vez no para rellenar dicha estructura, sino a través de dicha estructura, de manera que se "interpreta" en este momento el tráfico de entrenamiento en base al BloomFilter.

Los resultados que se obtienen al "interpretar" los ataques a través del Bloomfilter son ordenados, de manera que el primer elemento de esta lista corresponderá al N-Gram del ataque que menor puntución halla obtenido al ser "interpretado" por el Bloomfilter, es decir, aquel que apareció en menor medida en la etapa de entrenamiento del Bloomfiler.

De esta manera conseguimos determinar la distribución del tráfico que estamos utilizando en el entrenamiento (tráfico formado por ataques) en función del BloomFilter (construido con tráfico limpio) y, de esta manera, poder determinar qué valores disciernen en mayor medida entre el tráfico legítimo y el tráfico con ataques.

A la vista de los resultados obtenidos en las pruebas se decidió tomar como representativos los siguientes n-grams (que de ahora en adelante los llamaremos k_i): 1-gram, 2-gram, 4-gram, 8-gram, 16-gram, 32-gram, 64-gram y 128-gram.Estos valores han sido renombrados desde k_1 hasta k_8 respectivamente, para proporcionar mayor legibilidad.

Se ha optado por poner 128 como límite superior por razones de limitación tecnológica, puesto que valores mayores disminuyen la capacidad de análisis del sistema en tiempo real.

A continuación mostraremos un ejemplo en el que iremos indicando como se van rellenado las estructuras de nuestro sistema durante las diferentes etapas de las que consta el entrenamiento.

En primer lugar vamos a mostrar cómo se va rellenando la estructura BlommFilter (véase figura 7). Para simplificar, se considera un BloomFilter de 16 posiciones y n-grams de 4 bits (en lugar de ngrams de 3 bytes como se utilizan realmente), y a su vez, que el dataset de entrenamiento está compuesto únicamente por los 2 siguientes paquetes y que inicialmente el BloomFilterX está vacío.

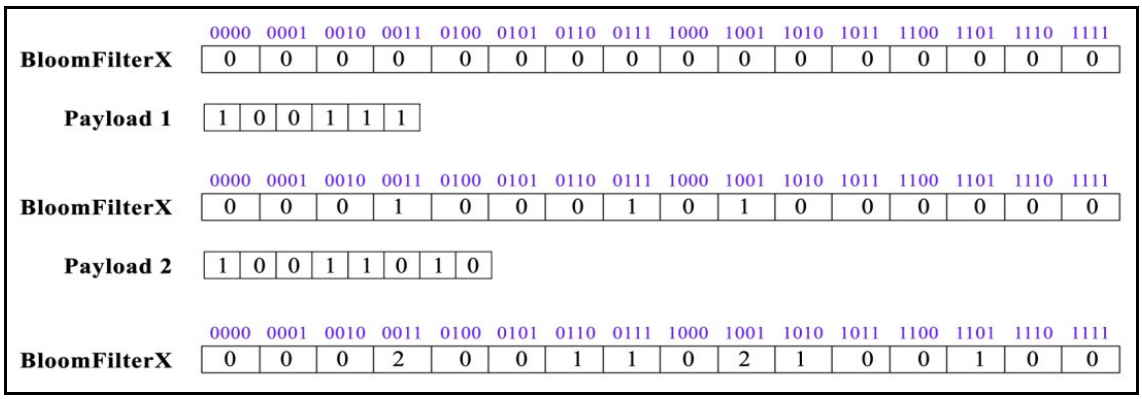


Fig. 7. Relleno del BloomFilter con tráfico legítimo

Imaginemos que ya hemos terminado la fase de Entrenamiento Bloomfilter y que el BloomFilter resultante de dicho entrenamiento es el que mostramos en la figura 7. El diagrama de la figura 8 explica cómo obtenemos las entradas que utilizamos para entrenamos la red neuronal SOM. Igualmente que en el ejemplo anterior, simplificamos utilizando un BloomFilter de 16 posiciones (que se ha seguido entrenando hasta que se ha mantenido "estable") y n-grams de 4 bits, teniendo solamente en cuenta también para simplificar únicamente K_1 y K_2 (en lugar de K_1 – K_8).

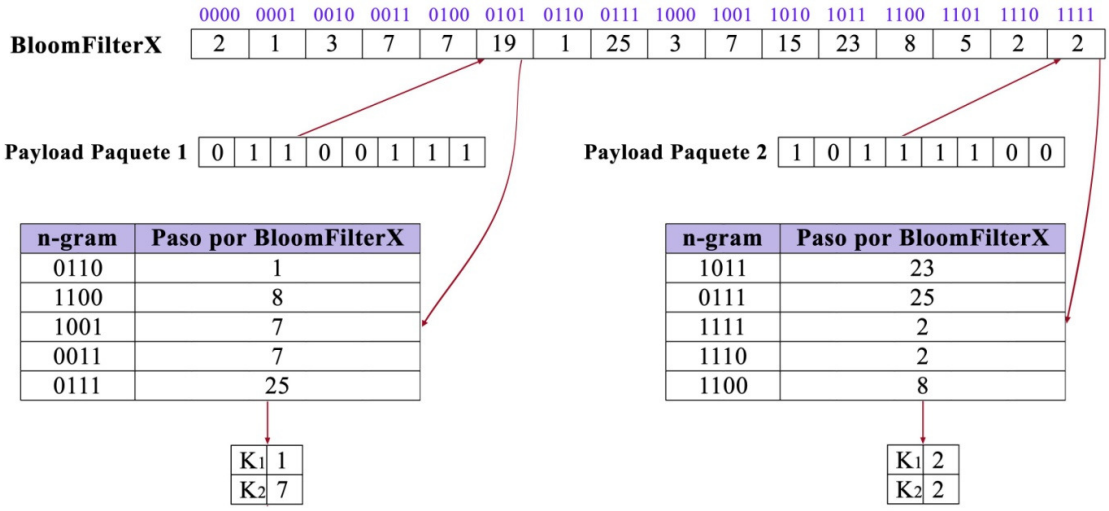


Fig. 8. Relleno de la Estructura de Referencias

NOTA. En el apartado de Resultados se muestran casos prácticos de este paso del entrenamiento, detallando los mapas topológicos que se obtienen al realizar el entrenamiento de la red neuronal SOM.

4.4.2.3. Etiquetado de los Cluster's

En esta etapa del entrenamiento realizamos un etiquetado automático de los cluster's creados por la red neuronal SOM en función de la proporción de paquetes de ataque y de tráfico limpio que contenga cada cluster o clase.

El procedimiento se divide en dos partes, en primer lugar, calculamos la proporción de ataques en comparación con el tráfico limpio que son clasificados por cada cluster de la red neuronal SOM.

En segundo lugar, etiquetamos cada cluster en función de los valores obtenidos en el paso anterior, utilizando cinco etiquetas distintas, crendo una escala que tiene en una extremo la etiqueta "probabilidad muy alta de ataque" y en el otro extremo la etiqueta "probabilidad muy baja de ataque" en función de la probabilidad de que dicho cluster identifique un ataque o tráfico limpio.

En la primera etapa de esta fase del entrenamiento, primero inicializamos un mapa bidimenional de 14x14 (análogo al generado por la red neuronal SOM) a 0

y para cada ataque que utilizamos en la fase de entrenamiento anterior tomamos todos los paquetes que lo componen y todos los cluster's que clasifiquen dichos paquetes se incrementan a razón de 1/(numero_de_ataques_utilizados). De esta manera damos un peso en el entrenamiento a los ataques compuestos por un número elevado de paquetes como a aquellos compuestos por un número reducido.

Una vez que hemos terminado este proceso para todos los ataques, realizamos un proceso similar con mapa análogo, pero en esta ocasión utilizaremos el tráfico limpio utilizado para entrenar nuestro bloomfilter, de manera que cada posición del mapa que corresponda a un paquete será incrementado a razón de 1/(número_de_paquetes_de_tráfico_limpio).

Finalmente, el mapa que utilizaremos para clasificar el tráfico de nuestra red será el resultante de dividir el primer mapa creado (tráfico de ataques) entre el segundo mapa creado (tráfico límpio).

Una vez que hemos construido dicho mapa procedemos a etiquetarlo. Para ello calculamos el valor máximo del mapa creado en el paso anterior y calculamos la raiz 6 de dicho valor (RAIZ). De esta manera cada una de las clases con las que etiquetaremos nuestro mapa de clasificación albergarán los siguientes valores:

- [0, RAIZ0) "1 – Probabilidad nula de ataque"
- [RAIZ0, RAIZ1) "2 – Probabilidad muy baja de ataque"
- [RAIZ1, RAIZ2) "3 – Probabilidad baja de ataque"
- [RAIZ2, RAIZ3) "4 – Probabilidad media de ataque"
- [RAIZ3, RAIZ4] "5 - Probabilidad alta de ataque"
- [RAIZ5, RAIZ6] "6 - Probabilidad muy alta de ataque"

- [RAIZ6, RAIZ7] “7 - Probabilidad total de ataque”

Es fácil observar que al seguir una progresión exponencial, la clase que contendrá mayor número de paquetes será la “1 – Probabilidad nula de ataque” mientras que la que contendrá menor número de paquetes será la “7 - Probabilidad total de ataque”.

4.4.2.4. Entrenamiento LVQ

En esta fase del entrenamiento utilizamos una Red Neuronal LVQ para realizar un “refinado” del etiquetado automático de los clusters creados por la Red Neuronal SOM.

(57) Las redes LVQ son uno de los algoritmos de clusterización no supervisada englobados dentro del área de las redes competitivas más utilizados en la actualidad.

Pese a que las redes SOM y LVQ se basan en los mismos fundamentos, presentan dos características diferenciadoras. En primer lugar, en las redes LVQ para determinados patrones de entrada podemos obtener más de una neurona vencedora, es decir, el mismo patrón de entrada puede clasificarse dentro de más de una categoría. En segundo lugar, mientras que la capa de salida de las redes SOM podía tener tantas dimensiones como necesitásemos (si bien lo habitual es utilizar redes de dos dimensiones) en las redes LVQ la capa de salida tiene una única dimensión.

Por ello, es habitual construir sistemas con ambas redes, tanto SOM como LVQ, trabajando de manera conjunta. (49) Las redes de tipo SOM se utilizan para el reconocimiento de patrones, de manera que cada patrón de entrada activa una única neurona de salida. Pero ahora nos encontramos con el problema de “clasificar” las neuronas de salida de la red SOM, es decir, dotar de un significado al mapa topológico formado por la red, puesto que la

intención de dicho mapa topológico es aproximar o "resumir" los valores de la señal de entrada disminuyendo, en la medida de lo posible, el error cometido.

Este es uno de los principales usos de las redes LVQ, puesto que con este tipo de redes contamos con la ventaja adicional de determinar si necesitamos una o varias neuronas vencedoras en la capa de salida para un determinado patrón de entrada, es decir, clasificar un patrón dentro de distintas categorías.

A grandes rasgos, una Red Neuronal LVQ es capaz, dada una lista de elementos etiquetados, de determinar si algún elemento de dicha lista no está correctamente etiquetado y proponer el etiquetado correcto. Para ello, básicamente calcula el centroide (punto más característico) de cada una de las clases en las que se divide la lista de elementos y posteriormente calcula la distancia de cada elemento de la lista con respecto a estos centroides.

Si el etiquetado es correcto, la distancia de un elemento con respecto a la de la clase con la que está etiquetado será menor que la distancia a cualesquiera del resto de los centroides. En caso contrario considera que un elemento ha sido etiquetado incorrectamente, y lo reetiqueta con el valor al que correspondería el centroide que se encuentra a menor distancia.

A continuación vamos a describir las principales características de nuestra red neuronal LVQ así como del entrenamiento utilizado para la misma. En primer lugar hay que aclarar, que al contrario que en el entrenamiento de la Red Neuronal SOM, hemos "personalizado" esta etapa del entrenamiento con respecto a lo que sería un entrenamiento "al uso" mediante una Red Neuronal SOM:

En lugar de utilizar un entrenamiento compuesto por un elevado número de ciclos, hemos implementado un entrenamiento compuesto por 5 iteraciones, de manera que en cada iteración se realiza el entrenamiento desde el principio pero a partir de los valores de la iteración anterior (en la primera iteración

tomaremos como punto de partida el etiquetado de los cluster´s realizado en la etapa anterior del entrenamiento). De esta manera evitamos en mayor medida caer en mínimos locales, en cuyo caso los resultados no sería tan satisfactorios como cabría esperar.

Los cluster's o elementos etiquetados como un extremo (“1- Probabilidad nula de ataque” y “7- Probabilidad total de ataque”) no permiten ser reetiquetados con un valor distinto, de esta manera forzamos que, en caso de ser necesario, el resto de etiquetas se adapten a estas clases. En caso contrario, en ocasiones, el número de clases se veía reducido, bien porque se reetiquetaban los elementos del extremo mayor o del extremo menor, de manera que espacio de clasificación se veía reducido.

Solamente se permite que una clase se reetiquete como una clase superior o inferior, por ejemplo, una clase de tipo 3 solamente se puede reclasificar como 2 o como 4 de esta manera también evitamos que el espacio de clasificación se reduzca.

El resto de características de la red neuronal LVQ son:

- Algoritmo de aprendizaje: learnlv2.
- Número de neuronas de la capa oculta: 30
- Número de salidas: 7.
- Nº de ciclos del entrenamiento: 100.
- Ratio de aprendizaje 0.05.

4.4.3. Fase de Detección

Una vez que se ha terminado el proceso de entrenamiento se pasa el tráfico a analizar por el sistema. Dicho tráfico se resume mediante el BloomFilter, la red

Neuronal SOM y la Red Neuronal LVQ de manera que obtenemos a qué clase pertenece el paquete que estamos analizando, es decir, si puede considerarse un ataque o no.

5. SIMULACIONES Y RESULTADOS

Para el entrenamiento y la comprobación de los resultados de nuestro sistema se ha elaborado un *dataset* (conjunto de *pcaps* representativos del tráfico y de los ataques conocidos) con la herramienta de captura de tráfico Wireshark.

Wireshark es una herramienta que permite ver el tráfico que pasa a través de una red mediante una interfaz gráfica muy intuitiva. Además tiene la ventaja de ser software libre, disponer de infinidad de manuales de uso y un gran reconocimiento en el mundo de la auditoría de sistemas. Por todo esto se decidió usar esta aplicación para capturar el tráfico que forma parte de los datasets utilizados.

Para realizar las distintas etapas del entrenamiento del sistema se necesitan crear varios conjuntos de datos: unos con tráfico limpio y otros con tráfico con código malicioso.

Para la creación de los datasets de tráfico limpio se capturó el tráfico de un ordenador doméstico durante varios días. Para poder crear un patrón de comportamiento fiable se realizaban todos los días acciones parecidas y visitas a las mismas páginas web. Las acciones capturadas son las siguientes:

- Tráfico generado por aplicaciones de intercambio de archivos P2P.
- Envío de correo electrónico mediante Hotmail con ficheros adjuntos (.doc, .pdf, .mp3, .jpeg...).
- Lectura de correo electrónico en Hotmail con ficheros adjuntos (.doc, .pdf, .mp3, .jpeg...)
- Navegación por páginas web de todo tipo (as, marca, mundo deportivo, diario sport, elpais, el mundo, abc, larazon, fdi ucm, ucm, upm, uam, forocoches, todoexpertos, yahoorespuestas, comunidad de Madrid,

gobierno de españa, the washington post, nba, universidad de Berkeley, wikipedia.es, wikipedia.org, cinetube, peliculasyonkis, seriesyonkis, vagos, etc.

- Descarga de programas y de aplicaciones del conocido servidor Softonic.
- Visualización de videos en youtube.

Para la captura del tráfico con ataques teníamos la necesidad de tener conjuntos de datos con un solo ataque aislado en cada .pcap, para poder entrenar y probar bien el sistema. Debido a esto decidimos descargar algunos virus y programas peligrosos conocidos de las siguientes páginas web:

- www.worst-viruses.2ya.com
- www.rigacci.org/comp/virus

Estos programas estaban comprimidos por lo que para poder capturarlos subimos los virus descomprimidos al conocido file hosting MegaUpload y capturamos la descarga de estos creando un archivo pcap por cada virus.

El listado completo de virus capturados se muestra en la tabla 9.

Backdoor.Buttman	I-Worm.Sobig.b	Parity.a
Backdoor.DonaldDick.15	I-Worm.Sobig.f	Parity.b
Backdoor.DonaldDick.152	I-Worm.Swen	PingPong.a
Backdoor.SdBot.aa	I-Worm.Tanatos.b	sasser.b
Backdoor.Zenmaster.102	I-Worm.Tanatos.dam	sobig
BitchSlap	I-Worm.Tettona	Stoned.a
Blaster	Joke.Win32.Errore	Trojan.JS.Seeker.o
Bloodlust	Joke.Win32.Zappa	Trojan.PSW.Hooker.24.h
Bomberman	JS.Fortnight.b	Trojan.W32.PWS.Prostor.A
Bye	JS.Trojan.Seeker.b	Trojan.Win32.DesktopPuzzle
Click 2	JS.Trojan.Seeker-based	Trojan.Win32.VirtualRoot
Die 3	Junkie.1027	Trojan.ZipDoubleExt-1
DnDdos	loveletter	Win32.FunLove.4070
Form.a	Macro.Office.Triplicate.c	Win32.HLLP.Hantaner
Gimp	Macro.Word.Cap	Win32.Xorala
happy99	Macro.Word97.Ethan	Win95.Dupator.1503
HDKP4	Macro.Word97.Marker.r	Worm.Bagle.AG
HLLC.Crawen.8306	Macro.Word97.Marker-based	Worm.Bagle.Z
IIS-Worm.CodeRed.a	Macro.Word97.Thus.aa	Worm.Mydoom.AS
IIS-Worm.CodeRed.c	melissa	Worm.Mytob.IV
I-Worm.Fizzer	MMR	Worm.Mytob.V.
I-Worm.Klez.e	MXZ II	Worm.P2P.SdDrop.c
I-Worm.LovGate.i	MXZ	Worm.SomeFool.Q
I-Worm.Mimail.a	NetBus 2.0 Pro	Worm.Win32.Fasong.a
I-Worm.Moodown.b	NetDevil 1.5	Worm.Win32.Lovesan.a
I-Worm.Mydoom.a	netsky.z	Worm.Win32.Muma.c
I-Worm.NetSky.d	nimda	Worm.Win32.Opasoft.a.pac
I-Worm.Rays	NYB	WYX.b
I-Worm.Sober.c	orm.Mytob.BM-2	Zip Monsta
I-Worm.Sober.c.dat	OwNeD	

Tabla 9. Listado de virus capturados

Este conjunto de virus se puede clasificar en las siguientes categorías:

- **Viruses:** Malware que tiene por objeto alterar el normal funcionamiento de la computadora, sin el permiso o el conocimiento del usuario.
- **Worms**: Malware que tiene la propiedad de duplicarse a sí mismo.
- **Nukers**: Nombre genérico para ataques de tipo DoS en TCP/IP.
- **Trojans**: Malware capaz de alojarse en computadoras y permitir el acceso a usuarios externos, a través de una red local o de Internet, con el fin de recabar información o controlar remotamente a la máquina anfitriona.
- **Macro viruses**: Virus elaborados mediante un macro lenguaje, por ejemplo, archivos .doc con código malicioso.

Finalmente, una vez descargados todos los códigos maliciosos y capturados en archivos pcap se dividieron en dos conjuntos de datos: uno para el entrenamiento y otro para realizar las pruebas de nuestro sistema y comprobar la eficiencia del mismo.

5.1. Simulaciones

En este apartado en primer lugar vamos a mostrar algunos resultados obtenidos durante el entrenamiento de manera que sirvan para profundizar en la comprensión del algoritmo implementado, haciendo especial hincapié en mostrar la distribución de clases dentro del mapa topológico generado tras el entrenamiento de la Red Neuronal SOM y en la comparativa entre el etiquetado de clases automático y el afinamiento obtenido de dicho etiquetado tras el entrenamiento de la Red Neuronal LVQ. Posteriormente mostraremos los resultados finales obtenidos en diversos experimentos realizados.

5.1.1. Clases obtenidas tras el entrenamiento de la red neurona SOM

En la figura 9 podemos ver la estructura de la red bidimensional que genera

como salida la Red Neuronal SOM. Este mapa topológico presenta una estructura hexadecimal, es decir, cada neurona tiene como vecinas a otras seis neuronas.

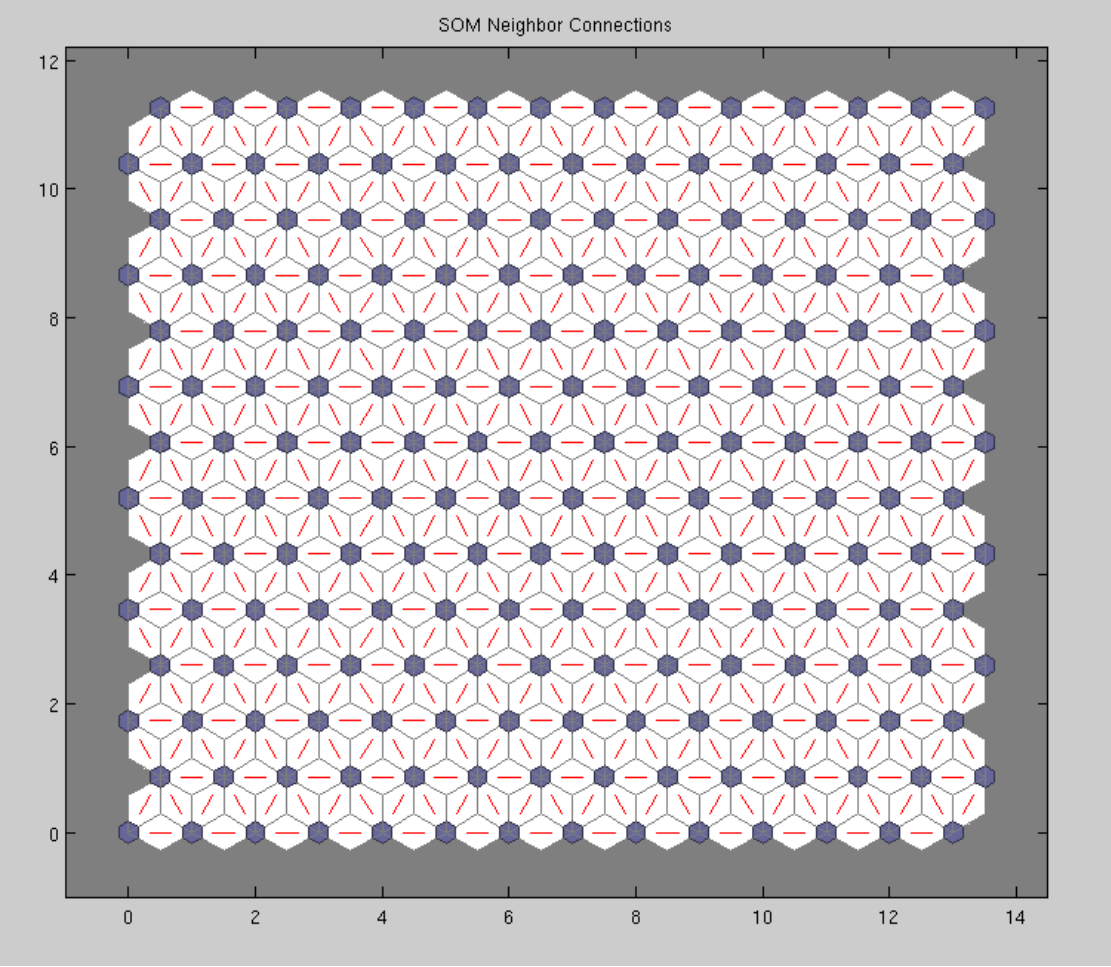


Fig. 9. Estructura de la red bidimensional

En la figura 10 podemos ver, una vez construida la red neuronal SOM, la distribución típica de las neuronas en cuanto a la distancia existente entre unas y otras. Hay que recordar que la red neuronal SOM no distribuye las neuronas de forma homogénea dentro de espacio de datos, sino que genera una mayor concentración de neuronas en aquellas zonas más densas, por lo que la distancia de unas neuronas a otras no tiene que ser contante, sino que dependerá de la propia distribución de los datos de entrada.

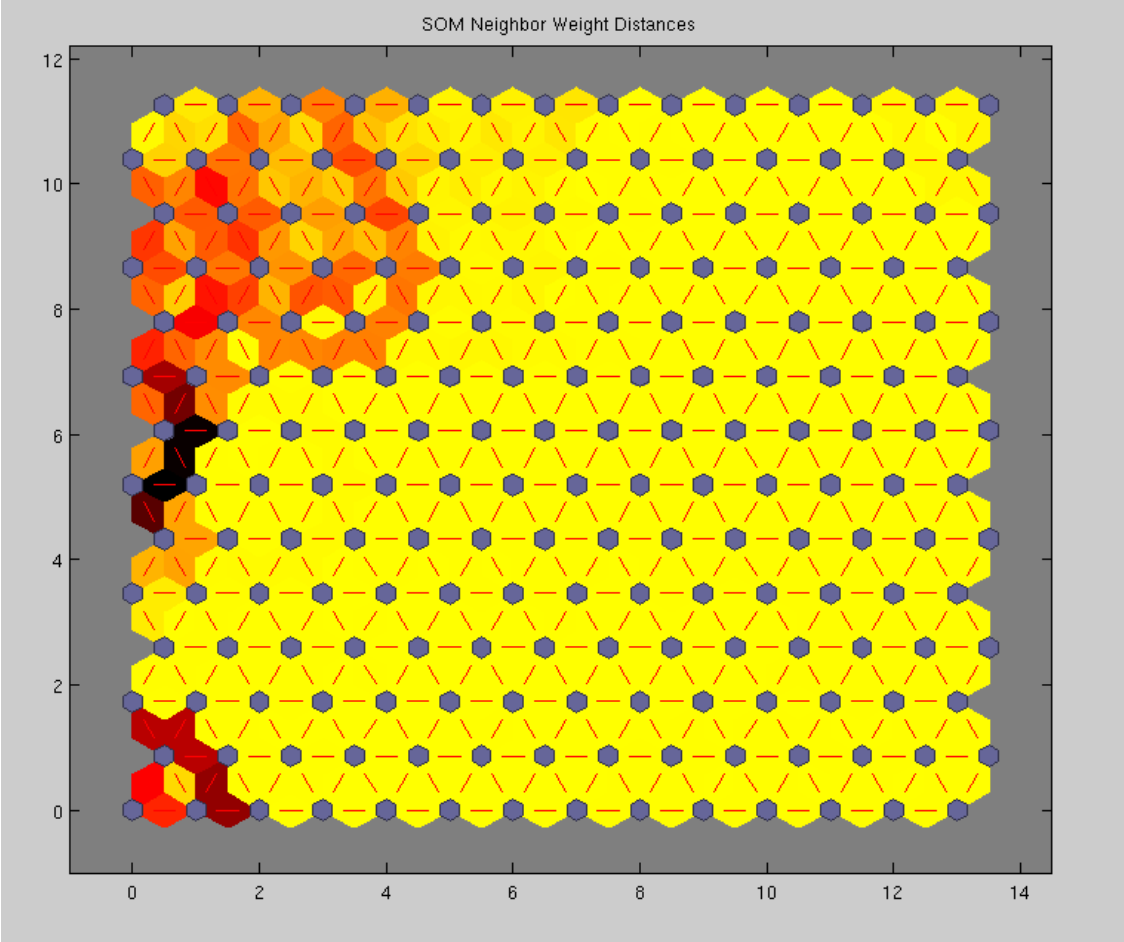


Fig. 10. Red Neuronal SOM

En la figura 11 podemos observar el peso que tiene cada una de las entradas en las neuronas que componen el mapa topológico de la red neuronal SOM. Lógicamente, la influencia de las entradas no es constante ni homogénea, sino que lo normal es que alguna entradas influyan de manera notable en determinadas neuronas mientras que en otras su influencia es casi inexistente y viceversa. Igualmente podemos observar como los mayores pesos corresponden a las zonas del mapa topológico que presentan las mayores distancias entre neuronas, las cuales corresponderán a las zonas de mayor "interés", puesto que serán aquellas que sirvan para discernir entre tráfico limpio y tráfico malicioso.

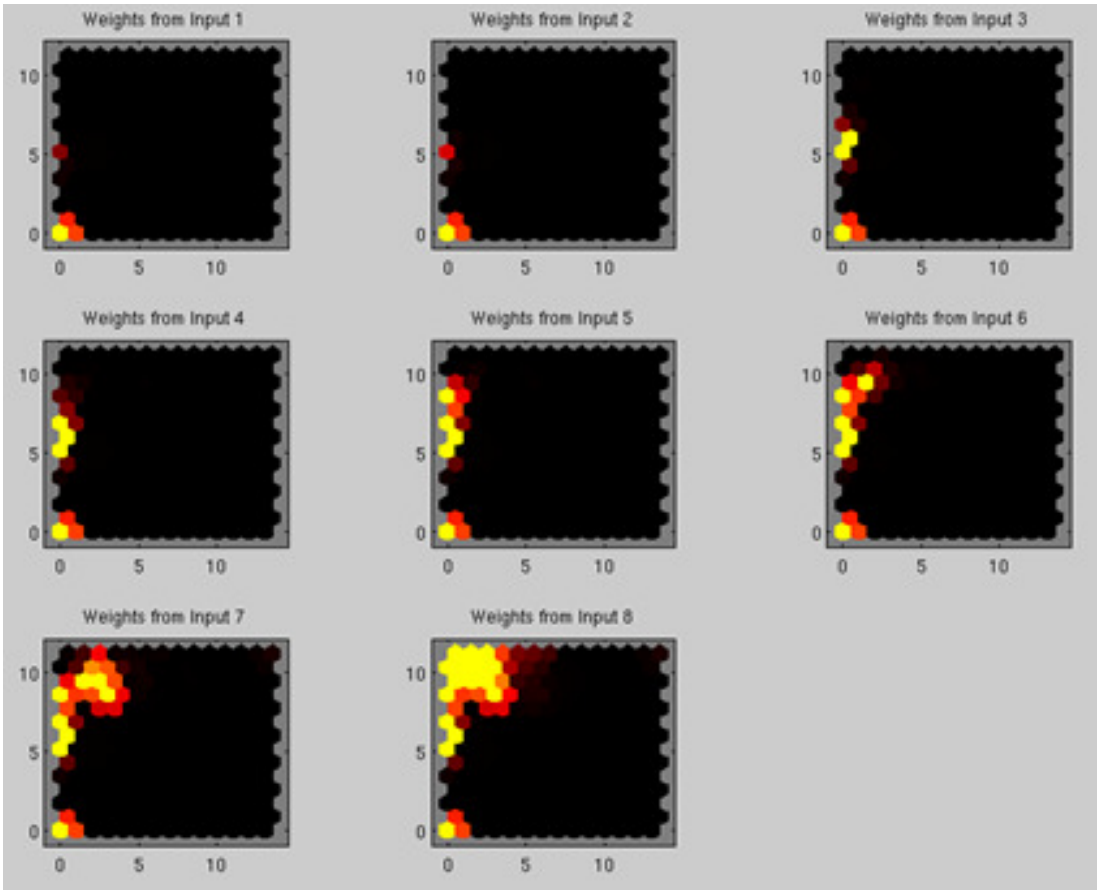


Fig. 11. Peso de cada entrada de las neuronas neuronas de la red neuronal SOM

Por último, en la figura 12 podemos observar cuántas entradas clasifica cada una de las neuronas del mapa topológico. Lógicamente esta distribución no es homogénea, de manera que las neuronas con mayor número de entradas corresponderán a aquellas que unívocamente corresponden a tráfico limpio.

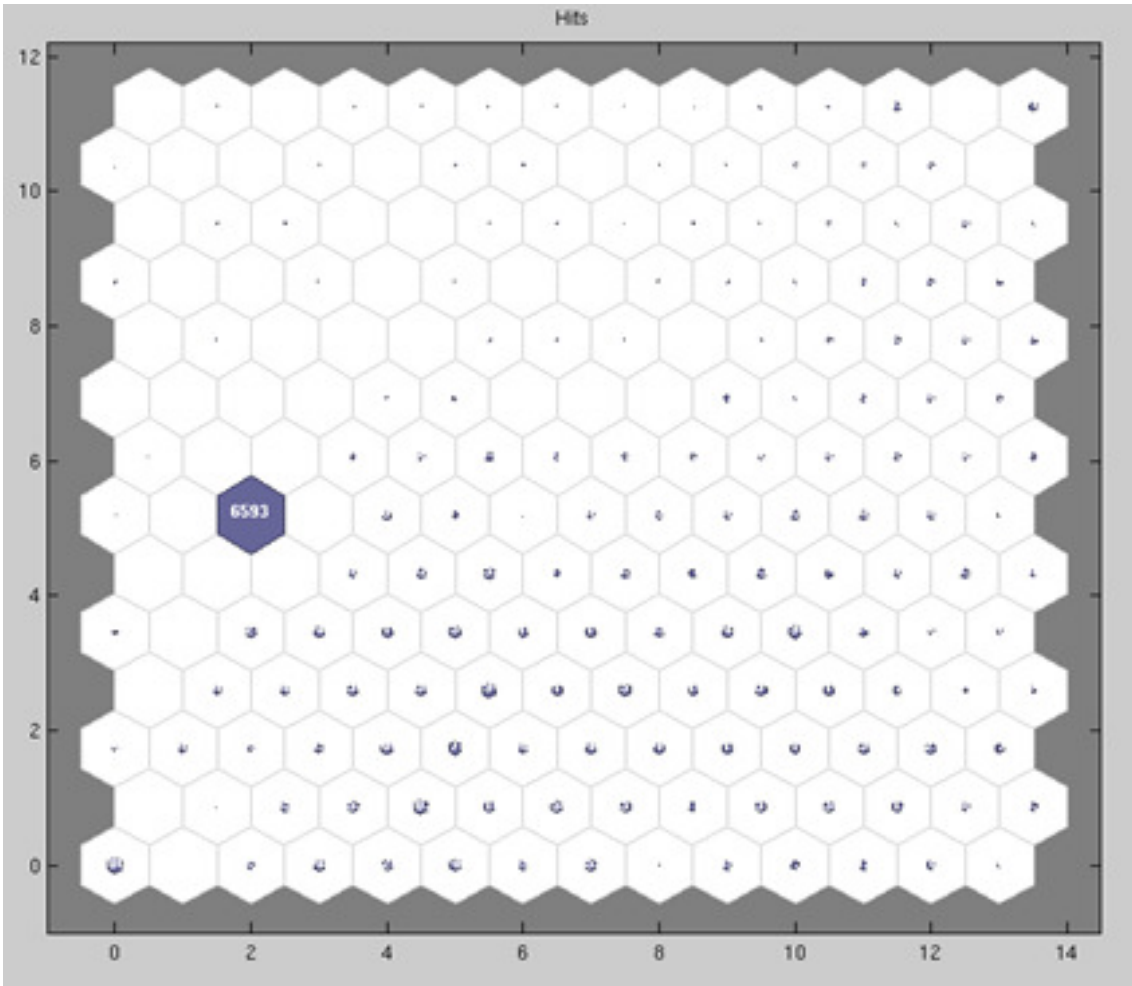


Fig. 12. Entradas que clasifica cada una de las neuronas del mapa topológico

5.1.2. Comparación de los resultados obtenidos con el etiquetado automático y con el etiquetado de la Red Neuronal LVQ.

En este apartado vamos a comparar los resultados obtenidos tras la clasificación automática de las clases generadas por la Red Neuronal SOM y el "afinamiento" que obtenemos tras aplicar la Red Neuronal LVQ a dicha clasificación.

Como podremos apreciar, el etiquetado automático es tan eficiente que si comparamos los resultados con los obtenidos tras aplicar la Red Neuronal SOM las diferencias son prácticamente inexistentes. En tablas 10, 11, 12 y 13 podemos

ver el número de falsos positivos generados, clasificados por clases, para cada uno de los experimentos realizados.

CLASE 1	CLASE 2	CLASE 3	CLASE 4	CLASE 5	CLASE 6	CLASE 7
57661	728765	27749	126869	0	410	6
59426	0	804295	64552	13025	156	6
74615	383852	296743	33884	0	7	6
56922	0	731816	0	0	363	6
159793	0	778676	4222	751	51	6
64320	0	767568	98521	13069	15	6
77960	453353	453133	465	8	250	6
21421	40670	687837	218659	0	16582	6
121952	584259	0	203692	298	783	6
87680	373531	422529	495	26726	23	6
86950	604090	0	286764	0	922	6
68266	465041	387405	57790	28	196	6
123835	0	723778	6120	0	368	6
28723	222381	543123	48959	0	10915	6
0	994507	9619	0	0	2333	1
159321	32509	785739	12490	11172	5223	6
0	758649	0	93105	0	308	6
137003	645900	25722	0	1761	0	6
146762	25526	629688	7742	667	1	6
79406	0	770227	32226	2667	45	6
140819	426172	249527	102	209	0	6

Tabla 10. Resultados de obtenidos con el etiquetado de la red neuronal LVQ

Como en la práctica las clase 1, 2 3 y 4 se pueden considerar sin lugar a dudas como tráfico limpio podemos combinarlas, obteniendo como resultado la tabla 11.

	CLASE 1-4	CLASE 5	CLASE6	CLASE 7
	941044	0	410	6
	928273	13025	156	6
	789094	0	7	6
	788738	0	363	6
	942691	751	51	6
	930409	13069	15	6
	984911	8	250	6
	968587	0	16582	6
	909903	298	783	6
	884235	26726	23	6
	977804	0	922	6
	978502	28	196	6
	853733	0	368	6
	843186	0	10915	6
	1004126	0	2333	1
	990059	11172	5223	6
	845573	3772	2720	3
	851754	0	308	6
	808625	1761	0	6
	809718	667	1	6
	881859	2667	45	6
	847550	25739	11282	6
	816620	209	0	6
	816407	0	422	6
MEDIANAS	**883047**	**118,5**	**335,5**	**6**
PORCENTAJE	**99,9479348**	**0,01341246**	**0,03797367**	**0,00067911**

Tabla 11. Resultados de obtenidos combinando las clases 1 a 4

CLASE 1	CLASE 2	CLASE 3	CLASE 4	CLASE 5	CLASE 6	CLASE 7
167687	614840	154410	4054	439	24	6
59426	454567	349634	77557	256	14	6
101008	488492	194181	4971	442	7	6
56922	337144	330349	64323	363	0	6
159793	521381	255337	6180	751	51	6
64320	437925	349077	79481	12685	5	6
77960	504207	356627	45652	669	54	6
21421	446789	281718	218492	16513	236	6
121952	584259	171698	32292	771	12	6
87680	373531	357181	92385	184	23	6
86950	604090	246811	39624	1108	143	6
68266	465041	332963	99347	12932	177	6
123835	427440	291386	9195	2199	46	6
28723	411782	333770	68718	11019	89	6
694606	129438	170463	9619	2205	128	1
159321	314723	420111	84185	22891	5223	6
112201	423771	301082	12291	2383	337	3
72468	408188	277993	92809	312	292	6
177009	605894	19663	6825	995	0	6
172288	467420	164485	5497	695	1	6
79406	605684	166736	31149	1551	45	6
64478	406594	308497	93720	10970	312	6
140819	510570	159798	5331	311	0	6
215351	267912	240366	92778	411	11	6

Tabla 12. Resultados de obtenidos con el etiquetado "automático"

Al igual que hicimos con los resultados del etiquetado de la Red Neuronal LVQ combinamos las clases 1, 2 3 y 4 en una única clase, obteniendo la tabla 13.

	CLASE 1-4	CLASE 5	CLASE 6	CLASE 7
	940991	439	24	6
	941184	256	14	6
	788652	442	7	6
	788738	363	0	6
	942691	751	51	6
	930803	12685	5	6
	984446	669	54	6
	968420	16513	236	6
	910201	771	12	6
	910777	184	23	6
	977475	1108	143	6
	965617	12932	177	6
	851856	2199	46	6
	842993	11019	89	6
	1004126	2205	128	1
	978340	22891	5223	6
	849345	2383	337	3
	851458	312	292	6
	809391	995	0	6
	809690	695	1	6
	882975	1551	45	6
	873289	10970	312	6
	816518	311	0	6
	816407	411	11	6
MEDIANAS	**896588**	**883**	**45,5**	**6**
PORCENTAJE	**99,89588**	**0,09838193**	**0,00506951**	**0,00066851**

Tabla 13. Resultados de obtenidos combinando las clases 1 a 4

Si utilizamos el Coeficiente de Correlación de Pearson para comparar los resultados "reducidos" (aquellos en los que hemos fusionado las clases 1, 2, 3 y 4 en una única clase) obtenemos un valor de 0,99985796, por lo que gracias a la Red Neuronal LVQ podemos concluir que el etiquetado de las clases de la Red Neuronal SOM generado automáticamente es correcto. De hecho, aunque para la clase 7 el número de falsos positivos es similar en ambos casos, para el resto de clases el etiquetado automático genera un número menor de falsos positivos que el obtenido mediante la Red Neuronal LVQ.

Sin embargo, el número de ataques detectados mediante ambas clasificaciones (como veremos en el apartado siguiente) es aproximadamente del 100%, siendo la única diferencia que para el mismo ataque la clasificación mediante la Red Neuronal LVQ genera un número de alertas ligeramente mayor, lo cual en algunas situaciones puede ser beneficioso puesto que a mayor número de alertas se puede considerar que mayor probabilidad existe de que nuestra red esté siendo víctima de una ataque, pero por contra, tenemos un mayor número de alertas a analizar que realmente se corresponden con el mismo ataque, con el gasto en recursos humanos que esto supone.

5.2. Resultados

En este apartado vamos a ver los resultados obtenidos, tanto para el algoritmo que utiliza el etiquetado automático como para el que utiliza el etiquetado generado por la Red Neuronal LVQ.

En la tabla 14 se muestra el número de alertas generadas al analizar el dataset de ataques en proporción al número de paquetes que componen dicho ataque utilizando el algoritmo de etiquetado automático.

	CLASE 1-4	CLASE 5	CLASE 6	CLASE 7
	0,94635316	0,0208432	0,00507428	0,02989653
	0,95856757	0,01249496	0,00126992	0,02705867
	0,9510554	0,02105051	0,00020067	0,02976501
	0,85949324	0,04478252	0,0459381	0,04489642
	0,94409962	0,02437321	0,00282668	0,02964703
	0,93666851	0,03400887	0,00061124	0,02690446
	0,92353518	0,04184676	0,00155077	0,0281936
	0,84071652	0,0379756	0,04571557	0,02629257
	0,94201571	0,02959127	8,3173E-05	0,02901074
	0,86937818	0,04445037	0,06268095	0,02826362
	0,81285623	0,04339844	0,0557204	0,02697274
	0,82407054	0,07249331	0,00120004	0,0287176
	0,89044426	0,10066091	0,00125767	0,02675818
	0,81838903	0,12532375	0,00699846	0,00421565
	0,9262845	0,05198835	0,05246302	0,02874447
	0,82152632	0,06442841	0,01748434	5,7786E-06
	0,9096044	0,02986979	0,05003407	0,02694799
	0,84548503	0,05591305	0	0,03066907
	0,91553703	0,0910339	0,00065083	0,03388651
	0,9071697	0,1080355	0,00394797	0,02812957
	0,80623028	0,09156981	0,00752455	0,02690593
	0,82427735	0,01031447	0	0,02912786
	0,96146778	0,02136114	0,04630774	0,03352453
PORCENTAJE	**0,9071697**	**0,04339844**	**0,00394797**	**0,02826362**

Tabla 14. Número de alertas generadas al analizar el dataset de ataques utilizando el etiquetado automático

De forma análoga en la tabla 15 se muestra el número de alertas generadas al analizar el dataset de ataques en proporción al número de paquetes que componen dicho ataque utilizando el etiquetado de la Red Neuronal LVQ.

	CLASE 1-4	CLASE 5	CLASE6	CLASE 7
	0,94644876	0	0,02507693	0,02847431
	0,90097227	0,06261181	0,0090087	0,02740722
	0,97133386	0	0,00019183	0,02847431
	0,85949324	0	0,11309955	0,02740722
	0,94409962	0,02464783	0,00277825	0,02847431
	0,89501702	0,07677043	0,00080533	0,02740722
	0,95791118	2,7562E-06	0,01361176	0,02847431
	0,86786957	0	0,10472321	0,02740722
	0,94083535	0,00118036	0,02950998	0,02847431
	0,84326716	0	0,12825853	0,02847431
	0,8688248	0,0006645	0,10310349	0,02740722
	0,963604	0	0,00792169	0,02847431
	0,82545066	0	0,14714213	0,02740722
	0,9262845	0	0,07364076	7,4744E-05
	0,86193623	0,01424744	0,09640912	0,02740722
	0,89847195	0,01113245	0,09039009	5,5123E-06
	0,87666363	0	0,09592916	0,02740722
	0,87560249	0,0959232	0	0,02847431
	0,95661483	0,01540329	0,00057466	0,02740722
	0,76777505	0,19964189	0,00410875	0,02847431
	0,75426597	0,07001138	0,14831543	0,02740722
	0,96453811	0,00698758	0	0,02847431
	0,86548997	0	0,10710281	0,02740722
PORCENTAJE	**0,89501702**	**0,0006645**	**0,02950998**	**0,02740722**

Tabla 15. Número de alertas generadas al analizar el dataset de ataques utilizando el etiquetado de la Red Neuronal LVQ

Si comparamos los resultados podemos observar que para los extremos (clase 7 o clase 1-4) los resultados son prácticamente idénticos tanto para el etiquetado automático como para el etiquetado generado por la red neuronal LVQ. Sin embargo, para las clases 5 y 6 los resultados difieren de manera notable, principalmente porque la red neuronal LVQ prácticamente "elimina" la categoría 5. Por esta razón se puede concluir con más satisfactorios los resultados obtenidos a través del etiquetado automático que los obtenidos a través de la red neuronal LVQ.

6. CONCLUSIONES Y TRABAJO FUTURO

Como características reseñables de este trabajo pueden señalarse las siguientes:

Al contrario de la mayoría de los trabajos realizados hasta la fecha sobre detección de código malicioso o malware, el sistema realiza dicha detección en el momento en que el sistema captura los paquetes de la red, mientras que la mayoría de los sistemas existentes realizan dicho análisis una vez que se ha recibido el binario que contiene el malware en su totalidad [22][34][35][37][40] o buscando comportamientos en el sistema que se salgan del estadísticamente típico del sistema mediante técnicas conocidas como cluster analysis [16][25].

También cabe destacar que el Sistema Inteligente de Detección de Intrusiones pertenece a los sistemas conocidos como "zero-day malware detection" de manera que es capaz de identificar nuevos "especímenes" de malware sin necesidad de depender de actualización externas.

Debido a la importancia de las fases de entrenamiento en este tipo de sistemas, se ha implementado un algoritmo que indica cuando puede darse por concluida la fase de entrenamiento de forma satisfactoria, lo cual supone una ayuda al administrador encargado del control de la red en la que se instale éste Sistema Inteligente de Detección de Intrusiones.

A su vez, al implementar el algoritmo como un preprocesador dinámico de Snort la instalación y configuración del mismo es mucho más eficiente y sencilla (en especial para aquellos administradores que estén familiarizados con el uso de Snort).

Conviene asimismo mencionar las principales dificultades encontradas:

En primer lugar, hubo que estudiar el diseño interno de un software tan complejo como es el de Snort, para poder desarrollar el sistema como un

preprocesador dinámico del mismo. Para ello fue de gran ayuda la plantilla que incluyen en la distribución de dicho software.

Sin embargo, el paso de convertir dicha plantilla en un preprocesador dínamico "con todas las de la ley", es decir, en igualdad de condiciones a las del resto de los preprocesadores dinámicos que se incluyen en la distribución de Snort fue un paso que requirió un gran esfuerzo, puesto que para tener éxito hubo que modificar diversos ficheros de dicha distribución para la inclusión del Sistema Inteligente de Detección de Intrusiones en Snort fuese satisfactoria. Aun así, ha merecido la pena, puesto que para aquellos administradores o consultores de seguridad que estén familiarizados con el uso de Snort la inclusión de nuestro sistema no supondrá un gran esfuerzo.

En segundo lugar, otro problema significativo que tuvo que resolverse fue encontrar un dataset adecuado para realizar el entrenamiento y posterior análisis.

Debido a la dificultad de encontrar un dataset que estuviera compuesto únicamente por envío de malware, se decidió finalmente recopilar un dataset a partir de malware obtenido de diversas páginas web "especializadas". El inconveniente de este hecho es lógicamente que las comparaciones con sistemas similares debe hacerse con las precauciones que supone realizar las pruebas con distintos dataset.

REFERENCIAS

[1] K. W. Salvatore, J. Stolfo: "Anomalous Payload-based Network Intrusion Detection", *Recent Advances in Intrusion Detection*, Springer, Vol. 3224, pp 203-222, 2004.

[2] D. Bolzoni, S. Etalle, P. Hartel, E. Zambon: "POSEIDON: a 2-tier Anomaly-Based Network Intrusion Detection System", *in Proceeding of the Fourth IEEE International Workshop on Information Assurance*, pp 144-156, 2006.

[3] R. Perdisci, G. Gu, W. Lee: "Combining Multiple One-Class Classifiers for Hardening Payload-based Anomaly Detection Systems", *in proceedings of Workshop on Machine Learning in Adversarial Environments for Computer Security*, 2007.

[4] S. A. Thorat, A. K. Khandelwal, B. Bruhadeshwar, K. Kishore: "Payload Content based Network Anomaly Detection", *Applications of Digital Information and Web Technologies* (ICADIWT 2008), pp 127-132, 2008.

[5] K. Wang , J. J. Parekh, S. J. Stolfo: "Anagram: A Content Anomaly Detector Resistant to Mimicry Attack", *Recent Advances in Intrusion Detection*, Springer, Vol. 4219, pp 226-248, 2006.

[6] V. A. Skormin, D. H. Summerville, J. S. Moronski, "Detecting Malicious Codes by the Presence of Their 'Gene of Self-replication'", *Computer Network Security*, Springer, Vol. 2776, pp 195-205, 2003.

[7] Y. Zhang, T. Li, J. Sun, R. Qin: "An FSM-Based Approach for Malicious Code Detection Using the Self-Relocation Gene", *Advanced Intelligent Computer Theories and applications, with Aspects of Theoretical and Methodological Issues*, Springer, Vol. 5226, pp 364-371, 2008.

[8] J. Zhang, M. Zulkernine: "Random-Forest-Based Network Intrusion Detection Systems", *IEEE Transation on Systems, man, and Cybernetics, Part C: Applications and Reviews*, Vol. 38, Issue 5, pp 649-659, 2008.

[9] J. Zhang, M. Zulkernine: "Anomaly Based Network Intrusion Detection with Unsupervised Outlier Detection", *in proceedings of IEEE International Conference on Communications (ICC ´06)*, Vol. 5, pp 2388-2393, 2006.

[10] H. Hung: "A Hybrid Network Intrusion detection Model Using Random Forest and K-nearest Neighbor", *in Technical Report: Master´s Thesis*, National Chung Cheng University, 2008.

[11] S. Bezobrazov, V. Golovko: "Neural Networks for Artificial Immune Systems: LVQ for Detectors Construction, Intelligent Data Acquisition and Advanced Computing Systems", *in proceeding of the 4th IEEE Workshop on Technology and Applications (IDAACS)*, pp. 180 – 184, 2007.

[12] M. F. Marhusin, D. Cornforth, H. Larkin: "Malicious Code Detection Architecture Inspired by Human Immune System", *in proceedings of the 2008 Ninth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*, Washington, DC, USA, pp. 312-317, 2008.

[13] J. Twycross , M. M. Williamson: "Implementing and testing a virus throttle", *in proceedings of the 12th conference on USENIX Security Symposium 12*, Berkeley, CA, USA, pp. 20-20, 2003.

[14] M. M. Williamson: "Throttling Viruses: Restricting propagation to defeat malicious mobile code", *in proceeding of the 18th Annual Computer Security Applications Conference (ACSAC)*, Las Vegas, NV, USA. pp. 61-68, 2002.

[15] J. Jung, V. Paxson, A. W. Berger, H. Balakrishnan: "Fast Portscan Detection Using Sequential Hypothesis Testing", *in proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, pp. 211-225, 2004.

[16] S. E. Schechter, J. Jung, A. W. Berger: "Fast Detection of Scanning Worm Infections", *in proceedings of the Seventh International Symposium on Recent Advances in Intrusion Detection (RAID)*, Sophia Antipolis, French Riviera, France, pp. 59-81, 2004.

[17] Y. Gu, A. McCallum, D. F. Towsley: "Detecting Anomalies in Network Traffic Using Maximum Entropy Estimation", *in proceedings of the Internet Measurment Conference*, pp. 345-350, 2005.

[18] M. V. Mahoney: "Network Traffic Anomaly Detection Based on Packet Bytes". *in proceedings of the 2003 ACM symposium on Applied computing (SAC)*, Melbourne, Florida, pp. 346-350, 2003.

[19] M. V. Mahoney, P. K. Chan: "PHAD: Packet Header Anomaly Detection for Identifying Hostile Network Traffic", *in technical report Florida Tech.*, 2001.

[20] D. Anderson, T. F. Lunt, H. Javitz, A. Tamaru, A. Valdes: "Detecting Unusual Program Behavior Using the Statistical Component of the Next-generation Intrusion Detection Expert System (NIDES)", in t*echnical report Computer Science Laboratory*, 1995.

[21] D. Anderson, T. Frivold, A. Valdes: "Next-generation intrusion detection expert system (NIDES): A summary", *in technical report SRI-CSL-95-07*, Computer Science Laboratory, SRI International, 1995.

[22] C.-Y. Lin, E. Hovy: "Automatic Evaluation of Summaries Using N-gram Co-Occurrence Statistics". *In proceedings of the Human Language Technology Conference (HLT-NAACL 2003)*, Edmonton, Canada, 2003.

[23] W. B. Cavnar, J. M. Trenkle: "N-gram based text categorization", *in proceedings of 3rd Annual Symposium on Document Analysis and Information Retrieval (SDAIR-94)*, Las Vegas, Nevada, pp. 161–175, 1994.

[24] A. Pagh , R. Pagh , S. Srinivasa Rao: "An optimal Bloom filter replacement", *in proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, Vancouver, British Columbia, 2005.

[25] F. Deng , D. Rafiei: "Approximately Detecting Duplicates for Streaming Data Using Stable Bloom Filters", *in proceedings of the 2006 ACM SIGMOD international conference on Management of data*, Chicago, Illinois, USA, 2006.

[26] F. Putze, P. Sanders, J. Singler, "Cache-, Hash- and Space-Efficient Bloom Filters", *in proceeding of the 6th International Workshop on Experimental Algorithms*, Springer Berlin / Heidelberg, pp. 108–121, 2007.

[27] A. Kirsch , M. Mitzenmacher, Less hashing: "same performance: building a better bloom filter", *in proceedings of the 14th conference on Annual European Symposium*, Zurich, Switzerland, pp.456-467, 2006.

[28] A. Broder, M. Mitzenmacher: "Network Applications of Bloom Filters: A Survey. A survey", *in proceedings of the 40th Conference on Communication, Control, and Computing*, University of Illinois: Urbana-Champaign, Illinois, 2002.

[29] P. S. Almeida, C. Baquero, N. Preguic¸a, D. Hutchison: "Scalable bloom filters", *Information Processing Letters*, Vol. 101 (6), pp. 255–261, 2007.

[30] B. Chazelle, J. Kilian, R. Rubinfeld, A. Tal: "The Bloomier filter: an efficient data structure for static support lookup tables", *in proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 30-39, 2004.

[31] Snort 2.1 Intrusion Detection (Second Edition), pp. 99-164, 2004.

[32] C. Schott, P. Wolfe, B. Hayes: Snort for Dummies, 2004

[33] Snort Users Manual 2.8.3, *The Snort Project.*

[34] A. F. A. Torres, C. E. B. Cortzar: "Snort Preprocessors Development Kickstart". 2005

[35] D. Ashley: "Developing a Snort Dynamic Preprocessor". 2008

[36] C. Parampalli, R. Sekar, R. Johnson: "A practical mimicry attack against powerful system-call monitor", *in proceedings of the Conference on Computer and Communications Security*, pp.255 – 264, 2002.

[37] T. K. Ho: "Random Decision Forests", *in proceedings of the Third International Conference on Document Analysis and Recognition*, pp. 278 -282, 1995.

[38] S. B. Medhdi, A. K. Tanwani, M. Farooq: "IMAD: In-Execution Malware Analysis and Detection", *Genetic and Evolutionary Computation Conference (GECCO)*, 2009.

[39] M. Z. Shafiq, S. M. Tabish, M. Farooq: "On the Appropriateness of Evolutionary Rule Learning Algorithms for Malware Detection", *in proceedings of the International Workshop on Learning Classifier Systems (IWLCS), held in conjunction with Genetic and Evolutionary Computation Conference (GECCO)*, 2009.

[40] S. M. Tabish, M. Z. Shafiq, M. Farooq: "Malware Detection using Statistical Analysis of Byte-Level File Content", *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), Workshop on CyberSecurity and Intelligentce Informatics (CSI)*, 2009.

[41] M. Z. Shafiq, S. M. Tabish, F. Mirza, M.Farooq: "PE-Miner: Mining Structural Information to Detect Malicious Executables in Realtime",

Recent Advances in Intrusion Detection, Springer, Vol. 5758, pp 121-141, 2009.

[42] M. Z. Shafiq, S. M. Tabish, M. Farooq, S. A. Khayam: "Embedded Malware Detection using Markovian Statistical Model of Benign Files", *Lecture Notes in Computer Science*, Springer, Vol. 5137, pp 88-107, 2008.

[43] M. Z. Shafiq, S. A. Khayam, M. Farooq: "Improving Accuary of Immune-inspired Malware Detectors by using Intelligent Features", *in proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 2008.

[44] M. Z. Shafiq, S. A. Khayam, M. Farooq: "Intelligent Features + Immune-Inspired Classifiers: An Improved Approach To Malware Detection", *in technical report of NexGin RC*, 2008.

[45] NexGin RC: "Artificial Immune System based General Purpose Intrusion Detection System (AISGPIDS)", *in technical report TR-nexGINRC-2009-21, FAST National University of Computer & Emerging Sciences*, g.

[46] NexGin RC: "A Bio-Inspired Self Defending Security Framework for IP Multimedia Subsystems (IMS-BISF)", *in technical report of National University of Computer & Emerging Sciences*, 2008.