Universidad de Deusto

Nuevo modelo de Detección / Prevención de Amenazas para la Seguridad Integral de los Sistemas de Información.

Tesis realizada por
Kenneth Lobato Lastra
para el grado de
doctor en filosofía

Dirigida por
Dr. Pablo García Bringas

Bilbao, Septiembre de 2015

Universidad de Deusto

Nuevo modelo de Detección / Prevención de Amenazas para la Seguridad Integral de los Sistemas de Información.

Tesis realizada por
Kenneth Lobato Lastra
para el grado de
doctor en filosofía

Dirigida por
Dr. Pablo García Bringas

Autor Director

Bilbao, Septiembre de 2015

A mis padres...

Abstract

Currently, in order to protect end-users and companies, Information Security is raising as an extremely important field. In this area, there are several traditional tools that have been working long ago, which provide subversive activities detection: Intrusion Detection Systems. IDSs mainly are built upon Misuse Detection or/and Anomaly Detection. Main issue of those tools is their reliability, having high False Positive / False Negative rates, mostly due to the huge volume of information they receive, not conceived when they were designed. Next Industrial step was the creation of more advanced systems, that could diagnose the threats with more accuracy and reduce the number of generated alerts, by means of joining/fusing evidences from several IDSs: Host, Network, etc. Fist evolution came with the creation of Collaborative Intrusion Detection Systems (CIDS), but in a short period of time, industry lead to the creation of SIEM systems, the-facto-standard. A SIEM worries about centralization of the diverse multi-heterogeneous-sources' evidences, but still based on not completely appropriate models (centralisation), architectures (hierarchical) and technologies (traditional SQL Databases - RDBMS). Present thesis outlines a paradigm change to take those Intrusion Detection systems to the next evolution: Big Data. Notwithstanding the huge volume of information to process, selected choice is to not face the computation using batches (MapReduce), but use real time flows: Streaming. In a controlled lab, where real traffic will be mixed with attacks over deployed vulnerable systems, a pilot is developed using the typical sources of an open-source SIEM system (OSSIM), demonstrating the computing performance of an scalable cluster of machines to process the evidences in real time. As secondary research line, over the previous infrastructure, clustering and classifying techniques will be applied over collected evidences, trying to look for knowledge extraction, capable of

detect correlations between the data in order to detect Zero Day Attacks, which is the real challenge that the Information Security sector is facing nowadays.

Resumen

El mundo de la Seguridad de la Información esta tomando una importancia significativa en la actualidad para proteger a los usuarios y a las empresas. Dentro de este área, existen una serie de herramientas tradicionales denominadas *Intrusion Detection Systems* que bien, basadas en detección de usos indebidos, bien basadas en detección de anomalías, llevan décadas funcionando y proporcionando una detección de actividades subversivas. El problema radica en que estas herramientas han dejado de ser completamente fiables, teniendo altas tasas de falsos positivos/negativos, ya que se enfrentan a un volumen de información inusitado en el momento en el que fueron diseñadas. El paso natural que dio la industria fue la creación de sistemas más avanzados, que aunando las evidencias de varios tipos de IDSs: Host, Red, etc., pudieran diagnosticar de forma más adecuada las amenazas, reduciendo el número de alertas que se generasen. La primera evolución fueron los Sistemas de Detección de Intrusiones Colaborativos, y finalmente el estándar de facto en las grandes compañías están siendo los sistemas SIEM, que se ocupan de centralizar las evidencias de múltiples fuentes heterogéneas. La pega es que se siguen basando en modelos (centralización), arquitecturas (jerárquicas) y tecnologías tradicionales (RDBMs). La presente tesis, plantea un cambio de paradigma para llevar a todos estos sistemas de detección de intrusiones al siguiente nivel evolutivo, el área de Big Data. A pesar de la cantidad de información que se tiene que procesar, la opción elegida es no abordar el procesado por lotes como en el caso de MapReduce, si no en flujos de datos en tiempo real: Streaming. Sobre la base de un entorno controlado dónde se realizarán ataques a máquinas vulnerables y con tráfico real, se despliega un piloto, denominado ESIDE-BIDS, que usando las fuentes de un sistema SIEM open-source (OSSIM), demostrará las capacidades de procesamiento en un cluster escalable de máquinas para analizar

las evidencias en tiempo real. Como segunda línea de trabajo, haciendo uso de la infraestructura anterior, se aplicarán técnicas de clusterización y de clasificación sobre las evidencias, en búsqueda de una extracción de conocimiento capaz de detectar correlaciones que ofrezcan la detección de ataques Zero-Day, el gran reto al que se enfrenta la Industria de la Seguridad de la información.

Agradecimientos

No puedo empezar la dedicación sin hacer una mención especial a mis padres, quienes siempre me han apoyado en todas mis metas a lo largo de los años, haciendo que sea la persona que actualmente soy. Desde pequeño me enseñaron a luchar por las cosas que se quieren en la vida, que el esfuerzo tiene su recompensa y que nadie suele regalar nada. A mi padre, agradecer su templanza, su saber hacer, su ingenio y su rigurosidad para todas las facetas de la vida. El esfuerzo que pongo en cada paso que doy se lo debo a él y los valores que me enseñó. A mi madre, agradecer su devoción, su inteligencia y la empatía que me enseñó a mantener con el resto del mundo, haciendo que pueda comprender y entender a cada persona que me encuentro en el camino. Gracias a ambos por el tiempo que me han dedicado a lo largo de estos años.

Partiendo de hace años, debo agradecer a mi director de tesis y amigo Dr. Pablo García Bringas, tanto la oportunidad que le dio a ese estudiante para incorporarse al equipo de Investigación de Seguridad (S3Lab), como el traslado de su vocación por la investigación, la matemática, la ingeniería y la aplicación de Inteligencia Artificial. Fueron esos primeros años de trabajar en la Universidad de Deusto, los que marcaron notoriamente el resto de vida profesional, integrado en un equipo multidisciplinar, que no tendría espacio en múltiples hojas de mencionar y agradecer su valía.

Fue de la mano de ese equipo de Seguridad donde comencé mi andadura en Panda Security, departamento de investigación (Panda-Research), con la Automatización en Entorno Real de Muestras de *MALWARE*. Hay una frase que me dijo el director de la división, Iñaki Urzay, que nunca podré olvidar: *"El Malware da dinero y por eso la gente desarrolla técnicas cada vez más inteligentes para conseguir robar a los demás"*. Debo agradecer a Iñaki

Urzay y Mónica Diaz, tanto su paciencia como su visión de la industria del Antivirus, que para una persona que viene del mundo académico se puede hacer complicado. Dentro de la misma Panda Security, aprendí un incontable número de tecnologías relativas al Área de Seguridad de la Información. Entre ellas el "reversing" y me resulta obligado mencionar a Saturnino Martinez, Senior Researcher, por sus enseñanzas. Así mismo, fue en estos días cuando se fue conformando la idea de la presente tesis, trabajando en Sistemas de Procesamiento Distribuidos y en cómo aplicar esta tecnología a soluciones reales de mercado. En esa época, ya estaba integrado en uno de los equipos más productivos de la compañía, y a todos ellos les debo felicitar y agradecer su apoyo: Javier Venero, Pedro Uría, Alejandro Montero, Luis Regel, Cecilia Gómez, Luis Corrons y demás compañeros. De igual forma, durante esa época fui vislumbrando la valía de la información y la potencialidad de las figuras de *Data Scientist*, tutelado en Bases de Datos por Carlos López López, gran profesional y amigo que me transmitió su pasión por los "datos".

En 2011, debido a la situación económica del mercado, cambié a un nuevo mundo de investigación: el centro de investigación IK4-IDEKO. Allí tuve que aprender otro sector y su investigación asociada: la industria de la fabricación de máquina-herramienta. Hasta antes de coger la excedencia laboral para finalizar la tesis, es dónde he estado trabajando, codo a codo con grandes compañeros, que a día de hoy me siguen apoyando en la realización de mis metas. En especial debo mencionar a: Jon Ander Ortiz, un genio en programación y resolución de problemas, que lleva conmigo desde la etapa en la Universidad de Deusto; a Gorka Unamuno, compañero y una de las personas que más me ha apoyado en que me pusiera a finalizar esta tesis. Por último, agradecer la comprensión de Rafael Lizarralde, mi jefe y Director de Investigación del Centro.

Finalmente, para concluir, debo agradecer a toda mi familia y amigos el apoyo que me han prestado en este tiempo, tanto para seguir con el día a día, como para sobrellevar temas familiares complicados que hemos debido de afrontar en el periodo de escritura de la presente tesis. ¡¡Muchas gracias a todos!!

Índice general

Índice de Figuras

Índice de Tablas

Índice de Código Fuente

« La invencibilidad es una cuestión de defensa, la vulnerabilidad, una cuestión de ataque. Mientras no hayas observado vulnerabilidades en el orden de batalla de los adversarios, oculta tu propia formación de ataque, y prepárate para ser invencible, con la finalidad de preservarte. Cuando los adversarios tienen órdenes de batalla vulnerables, es el momento de salir a atacarlos.»

El arte de la guerra. Sun Tzu - Siglo IV A.C.

Capítulo

1

Introducción

Hoy en día está en auge el término Seguridad de la Información. A diario se reciben noticias por los medios de comunicación de los peligros asociados a la Seguridad Informática. Se crea cada poco tiempo nuevo *Malware*[1] diseñado para distintos fines, entre los que se pueden encontrar: robo de credenciales, suplantación de identidad, ataques de denegación de servicio (DoS/DDoS) para forzar la no disponibilidad de servicios prestados, extorsión a usuarios finales y/o corporativos, intrusiones a compañías para robo de tecnología, ataques dirigidos entre distintos países para espiar, y un largo etcétera. La conciencia social comienza a valorar en mayor medida la necesidad de la protección de la información, reclamando soluciones más creativas y que ofrezcan mejor protección que los antivirus o los sistemas de detección de intrusiones tradicionales.

Según un análisis del fraude mundial de la compañía Kroll [Kro14], para los años 2013-2014, al menos el 70 % de las compañías han reportado ser víctimas de algún tipo de fraude, en comparación con el 61 % del año anterior, indicando en el reporte que el 21 % de las mismas se sienten vulnerables ante robos de información crítica, Figura 1.1.

Acorde a Trendmicro [Tre15], las tendencias para 2015 sobre Ciber Seguridad, se centran en la facilidad de creación del Malware actualmente

[1] *(ing. Malicious Software) Software cuyo objetivo principal es infiltrarse y/o dañar un sistema operativo sin el consentimiento de su propietario. Engloba otra serie de software malintencionado como por ejemplo: virus, troyanos, gusanos, rootkits, spyware, adware,...*

Chart 1. Percentage of companies affected by listed types of fraud

	2013	2012
Theft of physical assets	28%	24%
Information theft	22%	21%
Management conflict of interest	20%	14%
Vendor, supplier or procurement fraud	19%	12%
Internal financial fraud	16%	12%
Regulatory or compliance breach	16%	11%
Corruption and bribery	14%	11%
IP theft	11%	8%
Market collusion	8%	3%
Misappropriation of company funds*	8%	—
Money laundering	3%	1%

Figura 1.1: Kroll Report 2013-2014, Porcentaje de Compañías afectadas por Fraude.

con un potencial de peligrosidad suficiente para requerir atención, así mismo la introducción de terminales propios (*BYOD*[1]) supone un riesgo adicional por exposición de datos críticos. Finalmente el auge de *IoT*[2] puede conllevar nuevos vectores de ataque con peligro de integridad física de las personas (control de sistemas de refrigeración, apertura de barreras, etc.).

Siguiendo la línea de pronóstico anterior, la compañía Tripwire [Tri], apunta que: se ha visto un incremento del 2000 % en malware "evasivo", haciendo inoperativo el análisis estático, Figura 1.2; la autenticación basada en dos factores usada en terminales móviles, debido a la gran integración actual entre PC y móvil se ha visto comprometida; los ataques dirigidos se realizan en base a *exploits* personalizados que actualmente pueden ser automatizados en muchas de sus fases, generando un acceso a usuarios menos expertos con un peligro potencial muy elevado; la nube ofrece cada vez más servicios siendo objetivo complicado de proteger, dado que crece más rápido el ecosistema de aplicaciones que migran a la misma que la seguridad asociada.

Symantec, en su informe anual para 2013 - 2014 [Sym13], ofrece los siguientes datos significativos: tiene registradas un total 277 familias de malware para *Android*; una de cada 1.126 web-sites tiene malware; han aparecido 6.549 nuevas vulnerabilidades para Web; uno de cada 965 correos electrónicos en 2014 contiene suplantaciones (*phising - scams*); la media de URLs maliciosas para 2014 se fija en 3.829; los ataques dirigidos se centran

[1] *Bring Your Own Device.*
[2] *Internet Of Things.*

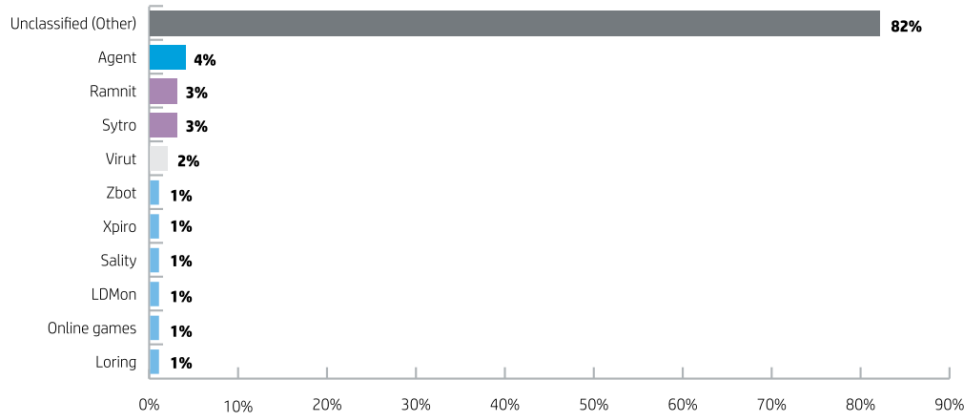


Figura 1.2: Muestras Top Malware descubiertas por ReversingLabs agrupadas por familia.

en compañías de *Manufacturing* con un 20 % del total. La distribución de *Zero Days* indicada por Symantec puede verse en la Figura 1.3, donde se ilustra el problema que puede llegar a provocar la falta de resolución de incidencias críticas sobre las que el *malware* puede basarse para realizar las acciones que le interese después de lanzar el correspondiente *"exploit"*. Así mismo, Symantec indica la aparición de 24.000 muestras nuevas en 2014 de media por día, frente a las 11.000 de 2013, en lo que se denomina como *Ransomware*, software que pide una cuantía económica para volver a dejar acceder a la información almacenada en la plataforma infectada.

SentinelOne [Sen15], indica que durante el 2014, el *malware* tipo *Ransonware* ha sido el más lucrativo, coincide con otras compañías de antivirus y soluciones de seguridad en el hecho del que el *malware* cada vez es más evasivo y dirigido específicamente, y sitúa a Windows como la plataforma más atacada. Entre sus predicciones para 2015, fijan como críticos los siguientes vectores para la siguiente generación de malware y ataques: (1) GNU/Linux empieza a representar un porcentaje importante dentro del despliegue de sistemas operativos mundiales e indican un posible incremento, que junto de la mano de Mac OS, ofrecerán nuevas vías de monetización para los atacantes; (2) creen que se desarrollarán nuevos softwares tipo "Ransomware" pero a nivel empresarial; (3) preveen ataques a infraestructuras críticas, como redes de alimentación eléctrica, basándose en ataques a sistemas SCADA, ICS y PLCs; (4) actualmente el MAAS (Malware As A Service) ha proliferado, si bien, establecen que en 2015-2016 se crearán plataformas de *Attacks As A Service* (AAAS).

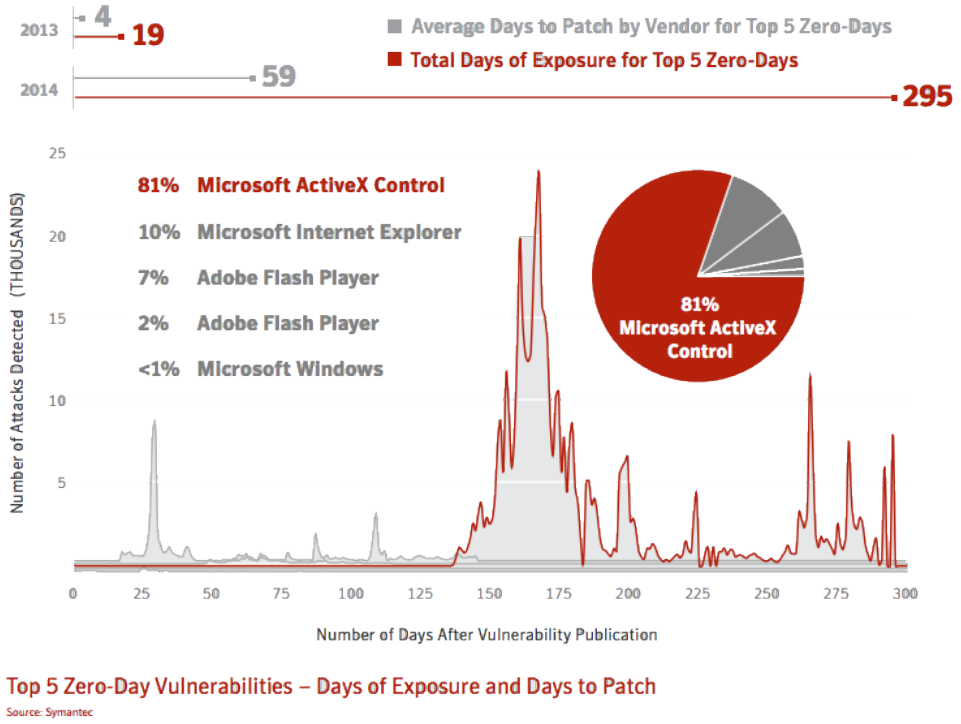


Figura 1.3: Zero-Day Vulnerabilities - Días de exposición y para subsanación mediante parche.

HP Security publica anualmente un reporte exhaustivo del año en curso, y los puntos claves que identifica son los siguientes:

- Los ataques se apoyan en código generado hace años y se observa que muchas de las vulnerabilidades explotadas durante el 2014 se han servido de código antiguo en una u otra etapa. Estos ataques han sido especialmente desplegados contra infraestructuras mal codificadas y/o configuradas en software tipo SAAS. La falta de configuración adecuada de servicios y servidores hace que muchas vulnerabilidades queden expuestas, Figura 1.4.

- Las nuevas tecnologías que se introducen a diario traen consigo nuevos vectores de ataque y retos en cuanto a seguridad. El primer *malware* orientado a dispositivos móviles se descubrió hace más de 10 años, si bien en 2014 ha sido el año en el que se ha extendido. Así mismo, la conexión de plataformas que antes no entraban en Internet, como los puntos de venta (PoS) son objetivos prioritarios para los atacantes.

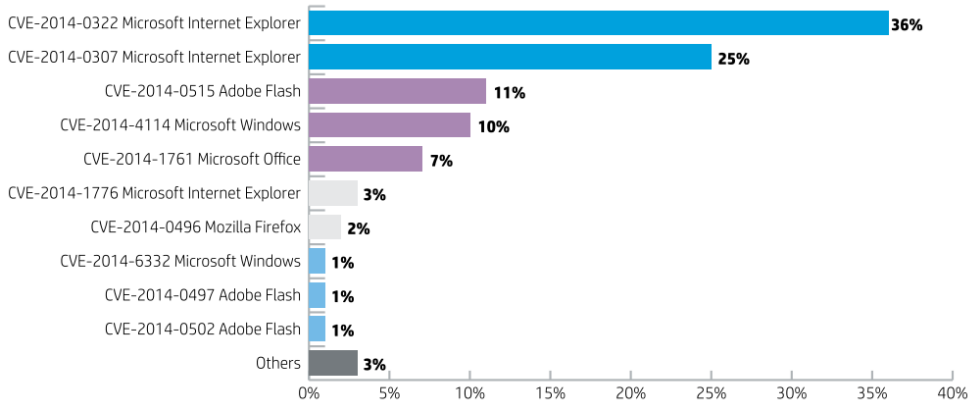


Figura 1.4: Muestras Top de Exploits en 2014.

- Se ha visto un incremento significativo en la visibilidad pública de ataques perpetrados por actores representando países. Adicionalmente a los países conocidos con este tipo de actividad, hay nuevos actores presentes, por ejemplo: Corea del Norte. El incremento de ataques desde China para ganar acceso remoto mediante troyanos (RATs) contra los manifestantes en Hong Kong, la intercepción del tráfico de Apple iCloud para conseguir nombres de usuario y contraseñas, así como el hackeo de la red TOR o el robo de datos masivos a Sony Pictures Entertainment, son ejemplos de que la seguridad es cada vez más visible y más necesaria.

- Se ha observado que la mayoría de las vulnerabilidades se basan en un número relativamente pequeño de errores de programación, que aunque se deba contar con la aparición de defectos en el software, el seguimiento de un proceso de desarrollo con visión de la capa de Seguridad puede minimizar y mitigar la frecuencia de la aparición de vulnerabilidades. HP Security Research, en conjunto con ReversingLabs, dispone de un catálogo de más de 100.000 exploits recogidos durante el año en curso, Figura 1.5.

- En Mayo de 2014, Brian Dye, vicepresidente de Symantec, declaró el antivirus muerto [Yad], si bien la industria del sector indicó justo lo contrario. Ambas percepciones son correctas, ya que un antivirus sólo identifica el 45 % de los ciber ataques. Es por ello por lo que los expertos de HP fomentan la combinación de varias técnicas de protección: usando todas las herramientas disponibles y no confiando en un único

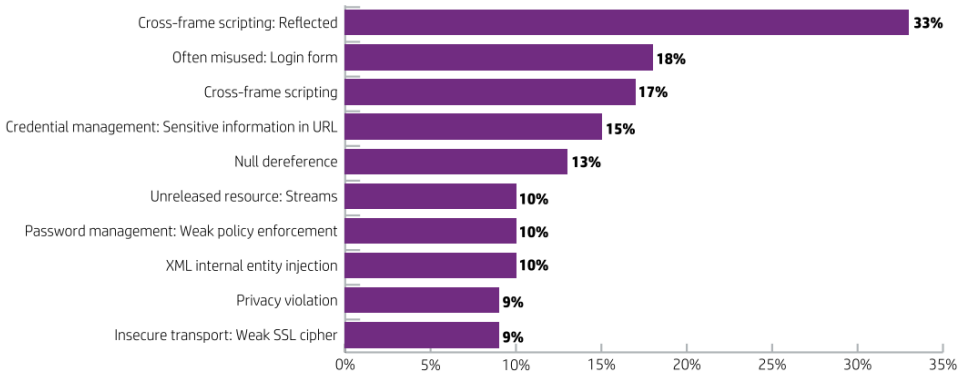


Figura 1.5: Top 10 Vulnerabilidades Críticas registradas en Aplicaciones Web durante 2014.

producto o servicio, los defensores pueden encontrarse en una mejor situación para prevenir, detectar y recuperarse de los ataques.

- La evolución del Malware a nivel mundial sigue prácticamente una ley exponencial de crecimiento, Figura 1.6.

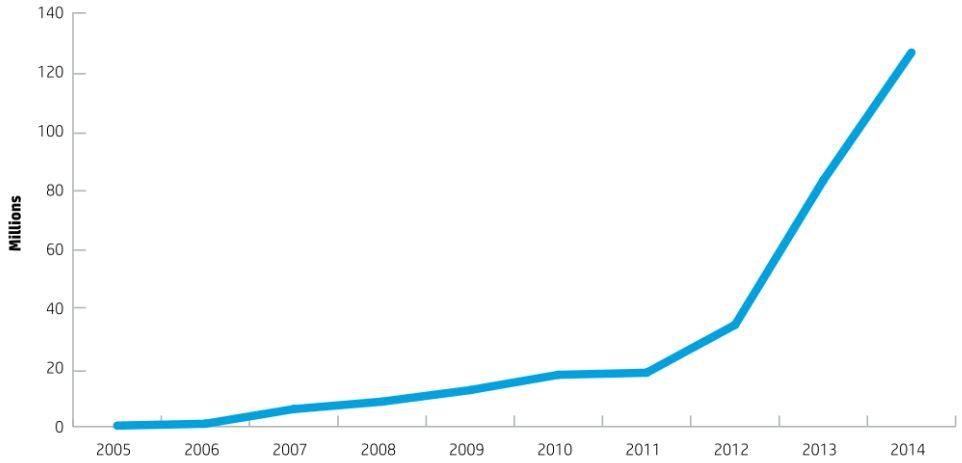


Figura 1.6: Muestras únicas de Malware recogidas por AV-Test.

Conforme el mundo avanza en la interconexión de los pobladores, el acceso a la información y a los servicios ofrecidos por Internet, se multiplica el riesgo de ser un usuario afectado por un problema de Seguridad. Analizando las amenazas, el mayor grupo de riesgo está en los usuarios que carecen de conocimientos técnicos en áreas de computación o de redes, pero que dedican una buena parte de su tiempo libre a éstas tecnologías. Así

mismo, por contra, el grupo con objetivos más rentables se ubica en la zona del tejido empresarial, donde la información confidencial de las nuevas líneas de negocio o de nuevos productos supone una ventaja efectiva sobre el resto del mercado.

Cada día más empresarios se dan cuenta del peligro al que se enfrentan y por ello se ha mejorado notablemente la seguridad, bien pasando por consultorías externas de expertos, bien contratando profesionales cualificados para salvaguarda de la información, o bien formando a los trabajadores e imponiendo políticas apropiadas de gestión del riesgo. Según varios reportes anuales las empresas a nivel mundial emplean el 5% de su presupuesto en la securización de la información. El 37% del total se destina a personal especializado y/o formación interna, el 25% al software de seguridad, el 20% al hardware, y finalmente en torno al 19% en servicios externos (outsourcing y consultoría)[Gar10, Ess10].

1.1 Situación Actual

A grandes rasgos si se analizan los diferentes tipos amenazas se pueden clasificar en dos grandes grupos no excluyentes: *network* y *host*. La diferencia en la clasificación radica en el foco de la propia intrusión. En el primer caso, el objetivo consiste en vulnerar los protocolos de red haciendo un uso incorrecto o indebido de ellos. En el segundo, un software malintencionado se apoya en vulnerabilidades del propio sistema operativo, o de las aplicaciones que tiene en ejecución, para realizar la intrusión-infección. No obstante, conforme ha ido evolucionando el sector de la Seguridad de la Información, ambos grupos se apoyan el uno en el otro:

- Existen ataques de red que vulneran protocolos para propiciar una brecha en los servicios software ofrecidos, pasando a ejecutarse dentro de la máquina en cuestión, comúnmente denominados *exploits*[1] remotos.
- Numeroso software malintencionado, *Malware*, tras infectar el Host, se vale de los protocolos de red para expandirse y propagarse infectando nuevas víctimas.

Una vez razonados los dos grandes grupos anteriores se entiende el hecho por el que la comunidad científica ha venido desarrollando dos líneas de trabajo, que se denominarán a partir de ahora como NIDS (*Network Intrusión Detection System*) y HIDS (*Host Intrusion Detection System*). La primera vertiente trata de localizar, y si es posible prevenir (IDS/IPS), un ataque en el momento en que se encuentra en la capa de red. La segunda se ocupa de detectar, bien mediante análisis estático, bien mediante análisis dinámico, un ejecutable malicioso dentro del sistema operativo o un comportamiento atípico del usuario/aplicaciones en el sistema. A su vez también se ha realizado un trabajo importante aunando ambas líneas de trabajo en lo que se denomina "*Hybrid Intrusion Detection Systems*" (HyIDS o HbIDS).

Adicionalmente, es conveniente señalar que en los últimos años, la mayor parte de las organizaciones han adoptado las nuevas tecnologías (*cloud computing*, *smartphones*, etc.), de forma que el grueso de los servicios informáticos adolecen del cambio, con el inherente peligro frente a las po-

[1] *Programa o técnica que aprovecha algún tipo de vulnerabilidad. Tienen dependencia directa con el sistema operativo en ejecución, los parches de seguridad aplicados, las aplicaciones en ejecución, la configuración de las mismas e incluso la arquitectura de red.*

sibles vulnerabilidades. La virtualización es uno de los cambios más importantes a la hora de los despliegues de las nuevas infraestructuras de computación, que hace que se compartan para diversos fines los recursos existentes: redes, servidores, capacidad de almacenamiento, aplicaciones y servicios. Uno de los mayores problemas derivados de este cambio es que los proveedores de las tecnologías de *cloud computing* deben proteger a sus usuarios y los datos sensibles que se almacenan. La recomendación de los expertos en Seguridad de la Información pasa por la combinación de las tecnologías existentes, haciendo especial énfasis en el uso de Sistemas de Detección de Intrusiones (IDS), que puedan monitorizar el tráfico de red y los cambios que se produzcan en el mismo, junto con la trazabilidad de configuración de los sistemas, el análisis de los ficheros de logs y las acciones realizadas por los usuarios finales [LKS10].

1.1.1 Nuevos paradigmas de computación.

La computación ubicua se definió conceptualmente por Mark Weiser, investigador de Xerox Palo Alto Research Center (PARC), siguiendo la inspiración de Philip K. Dick [Wei99] [DM11] como: *"la accesibilidad de los datos con oportunidades tecnológicas debe realizarse de forma continua e invisible"*. En el modelo *Ubik* de Philip todas las entidades objetivo se comunican entre ellas como un conjunto único de entidad inteligente. Las redes de comunicación actuales permiten el acceso y transferencia de los datos, de forma agnóstica a la localización de cada entidad.

El término *"Cloud Computing"* representa un modelo de comunicación interactiva que se lleva a cabo en más de un sitio de forma síncrona, fácil de usar, accesible siempre que el usuario la necesite, conformada por recursos de computación configurables y que requiere un esfuerzo de mantenimiento pequeño. A modo de resumen, cloud computing se puede definir hoy en día como la tecnología que permite la interconexión de servicios independientes temporal y espacialmente, que se ajusta a las necesidades del usuario final y que requiere un esfuerzo mínimo para mantenerse. Las 5 características fundamentales de Cloud Computing según Mell y Grace son [MG11]:

- **Servicio bajo demanda**: Sin intervención humana y/o del proveedor, el usuario dispone de capacidades de computación cuando sea necesario.

- **Accesibilidad de red global**: Sin importar el dispositivo desde el que se acceda, la tecnología debe proporcionar acceso desde cualquier punto.

- **Recursos Compartidos**: Los recursos computacionales: como el almacenamiento, la potencia de procesamiento, el ancho de banda de red o la memoria; pueden ser asignados dinámicamente o con dependencia a la demanda, pero siempre siendo de tenencia múltiple (*multitenant*). Los usuarios no necesitan de ninguna autoridad sobre los recursos y desconocen dónde se encuentran los mismos.

- **Elasticidad**. Las capacidades de los servicios prestados deben ser escaladas de la forma que sea, tanto ampliando recursos como reduciendo los mismos según las necesidades.

- **Servicio regular**: Así como los servicios prestados en *Cloud Computing* no pueden ser regulares ya que cambian con el tiempo y la demanda, la infraestructura de recursos sobre la que corren sí puede ser monitorizada, controlada y reportada convenientemente, de forma que el uso de los recursos pueda ser determinado y los proveedores puedan ofrecer los mismos a los usuarios de forma transparente.

Según el NIST se definen 5 actores principales dentro de la infraestructura de Cloud Computing [20113]:

- **Consumer**: El usuario del servicio de cloud computing, que puede ser un individuo o una unidad organizativa.

- **Provider**: Entidad responsable del desarrollo de los servicios y de proporcionar los recursos. Los proveedores gestionan el software y la plataforma, incluyendo los niveles de servicio pactados de confianza, seguridad, disponibilidad, etc.

- **Auditor**: Actor que se ocupa de inspeccionar el flujo de la información, el rendimiento y la seguridad asociada. Generalmente este actor deberá ser una tercera entidad, no perteneciente al grupo de consumidor y/o de proveedor.

- **Broker**: Figura que se ocupa de la gestión de la conexión entre el proveedor y el consumidor, gestionando el rendimiento y la disponibilidad del sistema.

- **Carrier**: Se ocupa de realizar la conexión, comunicación y transferencia entre proveedor y consumidor

Sobre los servicios que se prestan en Cloud Computing hay una definición que sigue la regla "Anything As A Service" (XaaS). La X define el servicio en cuestión, por ejemplo: CaaS, Comunicación como servicio; NaaS, red de comunicaciones como servicio; MaaS, monitorización como servicio; y un largo etcétera. De todas formas, los tres servicios principales que se suelen ofrecer son: **IaaS**, infraestructura como servicio; **PaaS**, plataforma como servicio; y **SaaS**, software como servicio. En el primero de los servicios, el de infraestructura, los usuarios tienen acceso a procesadores, unidades de almacenamiento, ancho de banda de red y otros recursos compartidos, que incluyen sistemas operativos y aplicaciones. Si bien en IaaS, los usuarios finales no son responsables del control y gestión de la infraestructura cloud física, ya que disponen de autoridad a partir del sistema operativo que despliegen. En cambio, en PaaS, los usuarios desarrollan y corren software sobre la infraestructura cloud a través de lenguajes de programación, librerías y servicios proporcionados por el proveedor. En PaaS, los usuarios no tienen responsabilidad de controlar y manejar la red, los servidores, el sistema operativo o los medios de almacenamiento. Finalmente, SaaS representa que toda la infraestructura, plataforma y utilidades software son proporcionados por el proveedor. En SaaS los usuarios acceden a las aplicaciones servicio vía distintos dispositivos e interfaces como clientes ligeros, teniendo una capacidad de configuración muy limitada. La Figura 1.7 muestra un esquema de los modelos y los actores en *Cloud Computing*.

1.1.2 Seguridad de la Información: Tradición y Futuro.

Mientras se pasa del paradigma de la computación tradicional a los nuevos paradigmas de computación: *cloud computing*, *big data*, *smartphones*, etc.; se deben afrontar nuevos retos dentro del área de la Seguridad y la Privacidad. Siguiendo la línea tradicional el primer punto a abordar es la *Seguridad Física*, que hace referencia a las propiedades físicas. Por ejemplo, dentro de un Centro de Datos que es propiedad del *Provider*, se deben de seguir una serie de estándares para mantener la seguridad globalmente: certificaciones, supervisión y mantenimiento de medidas preventivas de seguridad, fuentes de alimentación ininterrumpidas, continuidad de negocio (terremotos, inundaciones, incendios, etc.) y seguridad de acceso perimetral.

Dentro del marco tradicional de la *Ciber Seguridad*, las nuevas infraestructuras proporcionan un nuevo sector a proteger. Los apartados más

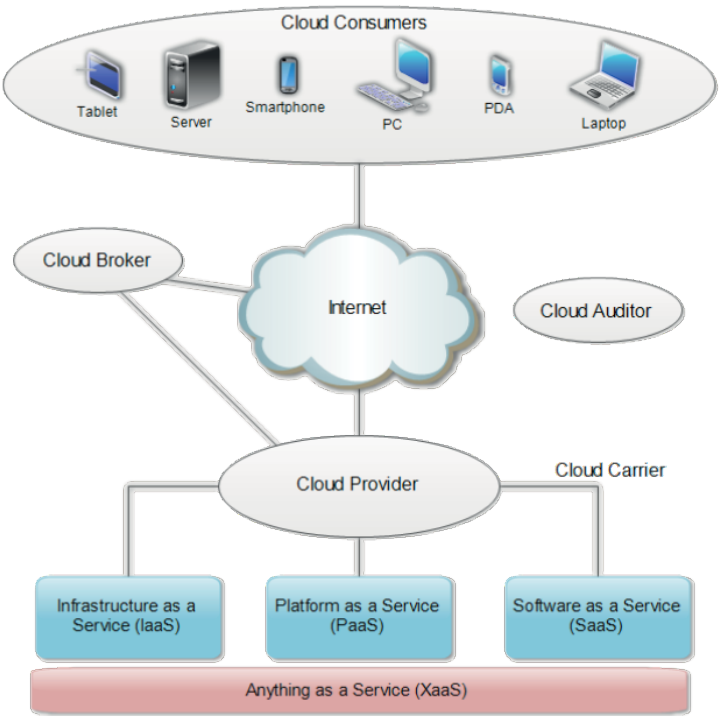


Figura 1.7: Modelos de Servicio y Actores implicados en Cloud Computing.

comunes a los que deben hacer frente los expertos en seguridad son los siguientes:

- **Ataques desde dentro**: Se define como *Insider* a los empleados, emprendedores y asociados que pueden tener acceso a la infraestructura con una autoridad privilegiada [CT09]. Los ataques desde dentro se organizan y ejecutan por los individuos anteriores para dañar o extraer datos críticos de los clientes o proveedores y pueden incluir todo tipo de ataques desde el interior haciendo uso de sus privilegios en la infraestructura [DCG12] [MPB+13].

- **Ataques de denegación de servicio**: Este tipo de ataques, denominados *Flooding* en la literatura, consiste en enviar grandes cantidades de paquetes TCP, UDP, ICMP u otros protocolos de red en solitario o en combinación, de forma que se puedan usar ordenadores en modo *Zombie* [MPB+13]. Las nuevas infraestructuras dado que son accesibles desde la Internet Pública hace que sean susceptibles a ataques de

denegación de servicio (DoS [1]) o de denegación de servicio distribuido (DDoS[2]). Un ataque *DoS* de este tipo que se realiza a un servidor que sirve un determinado servicio, conlleva un problema de accesibilidad al mismo para los usuarios finales, que se considera *Indirect DoS* cuando además el servidor colapsado en cuestión proporciona otros servicios.

- **Ataques de elevación de privilegios**: En este tipo de ataques, el intruso trata de conseguir unos permisos o autorizaciones que no dispone de partida en la infraestructura o servicio. Generalmente se hace uso de técnicas como: *Buffer Overflow*, *String Formating*, *Race conditions*, *etc.*; que no dejan de ser explotación de vulnerabilidades en el software que está corriendo, con el objetivo claro de: (1) conseguir ejecutar otro software con privilegios diferentes, (2) conseguir información sobre las cuentas de usuarios autorizados en el sistema. Los fines últimos de los intrusos que usan este tipo de técnicas pueden ser muy diversos, siendo el más común el robo de información [Eri08, AHLR07].

- **Escaneo de puertos**: Como primera medida que adoptan los intrusos a la hora de examinar una infraestructura a atacar, suelen dedicar un tiempo a recabar información de la misma. Para ello se identifican los puertos de comunicaciones que están abiertos, cerrados o filtrados. En el caso de puertos abiertos, o filtrados, se puede conectar con los servicios de forma que se obtenga más información: versión del software que corren, límites, fallos en la configuración de los mismos, etc. El escaneo de puertos suele ser la primera medida que se realiza por un intruso durante las fases previas a la intrusión y se tiende a englobar en la literatura como: *Recogida de Información* [LF09, Spa15, DGBJ14, MS06].

- **Ataques a la virtualización**: Los sistemas de virtualización se basan en los denominados *hipervisores*, que no es más que el *kernel* que controla las máquinas virtuales. Un ataque de este tipo, donde se consigue tener acceso al *hipervisor* ,conlleva una exposición de todas las máquinas virtuales que se encuentren en el servidor o *cluster* de servidores. En este ámbito toman especial relevancia los ataques denominados como *Zero Day*, que no son más que vulnerabilidades antes de que el fabricante del software tenga conocimiento de su existencia y/o

[1] *DoS:* Denial of Service.
[2] *DDoS*: Distributed Denial of Service.

no se encuentre solución en el mercado vía parches o actualizaciones [RIAH11].

- **Ataques de puerta trasera**: Un ejemplo de este tipo de ataques puede ser un ataque de tipo pasivo en el que los intrusos comprometen un nodo dentro de la nube y lo usan a modo de *Zombie* para ejecutar ataques tipo *DDoS*. Así mismo, la vulneración mediante *Trojanos* de un nodo concreto puede ayudar a comprometer la infraestructura al completo [MPB+13].

- **Capacidad de almacenamiento**: En las nuevas infraestructuras y grandes centros de almacenamiento de datos, actualmente existen una serie de riesgos elevados de cada a cómo se procesa la información, quién tiene acceso a la misma y cómo se realizan los procesos de auditoría y mantenimiento, así como qué parte de la misma será accesible por los usuarios finales. Se debe garantizar desde el área de Seguridad de la Información que los datos son privados, están debidamente aislados entre usuarios y se deben cumplir los compromisos de servicio [SSR+12].

- **Gestión de la Identidad, Autorización, Autenticación y Cifrado**: Con diferencia de las tecnologías convencionales, dado el uso intensivo de las máquinas virtuales en los nuevos sistemas de computación, los recursos son dinámicos y es por ello por lo que toma un especial interés el garantizar la Identidad del usuario, la Autorización que tiene para acceder a las funcionalidades y la protección de la información mediante cifrado ante posibles brechas [SSR+12].

- **Integridad de la información**: Los proveedores que tienen nivel de confianza suficiente o los mismos administradores de sistemas pueden manipular los datos para sus propios beneficios. Con la correspondiente combinación de cifrado y firmas resumen[1] se pueden conseguir resultados adecuados para garantizar que los datos no han sido modificados, como por ejemplo las cadenas de protección de *AdjointVM* [MDV13, Kon11, OAS13].

Dentro del área de trabajo para para detectar, prevenir o por lo menos mitigar los ataques descritos anteriormente se encuentran los **Sistemas de**

[1] *Hashes*: Algoritmo matemático para extraer una firma única de un conjunto de datos. Ejemplos: MD5, SHA, ...

Detección de Intrusiones (*IDS*[1]). Los IDSs pueden ser Software o Hardware y se debe dejar claro que no todo evento sospechoso que indiquen es una intrusión, punto clave para definir lo que es anómalo pero sin peligro, frente a lo que realmente tiene riesgo. La clasificación de este tipo de sistemas, principalmente, tiene los siguientes grandes grupos:

- ***Host Based IDS***: Los HIDS se ocupan de analizar las actividades sospechosas en la Configuración del sistema, en los procesos e hilos que está ejecutando el Software, almacenamiento y gestión de ficheros, así como llamadas de bajo nivel que se realizan al **kernel**. Los HIDS también pueden observar el tráfico de red, si bien desde una perspectiva del huésped en el que están instalados, con la pega de si están a un nivel suficientemente bajo como para que un *Rootkit* no afecte a su visión. Primordialmente se basan en la monitorización de los recursos del sistema en el que se instalan, funcionando como agentes que reportan cuando detectan una actividad sospechosa.

- ***Network Based IDS***: Los NIDS se ocupa de observar, monitorizar y analizar el tráfico de red. Se suelen emplazar en los puntos de acceso entre redes privadas e Internet, bien ubicados en la DMZ o bien en el router que hace las veces de *gateway* con el exterior, generalmente Internet. Los NIDS se ocupan de analizar el tráfico de red en busca de evidencias de actividades maliciosas.

- ***Hybrid IDS o Distributed IDS***: Los HbIDS o DIDS son los sistemas con los que se realiza la detección de intrusiones en entornos distribuidos como *GRID* y *Cloud Computing*. Básicamente es como juntar NIDS con HIDS, de forma que cada uno de los Agentes cooperen reportando a su nivel superior.

- ***Network Behavior Analysis Intrusion Detection***: Estos sistemas se ocupan de modelar el tráfico de red en base a parámetros estadísticos, de forma que los sensores puedan detectar ataques tipo *DoS* o propagación de *Worms* [2].

Si se analiza más profundamente las técnicas que se emplean habitualmente en la detección de Malware para entorno Host o en aquellas usadas

[1] *IDS*: Intrusion Detection System

[2] *Worm* o *Gusano*: Programa informático que puede correr de forma independiente y puede programar una versión completa de sí mismo a otros nodos de la red, pudiendo consumir recursos de los nodos infectados de forma destructiva.

para detección de intrusiones en entorno Network, se observa una correspondencia en las metodologías empleadas:

1. Basados puramente en conocimiento experto:

   (a) Entorno de Máquina o *Host*:

      i. Detección estática. Esta vertiente se basa en el conocimiento previo del ejecutable en cuestión. Generalmente se analiza bien por personal experto, bien por sistemas automáticos en busca de características relevantes que permitan un reconocimiento. Para ello se pueden utilizar funciones resumen (MD5, SHA,...) de la muestra al completo o bien centrarse en determinadas partes del binario en cuestión para identificar la muestra (hashes de diferentes partes, detección basada en reglas, etc.). Si se amplía la firma de detección se habla de firmas genéricas para detección de familias de Malware (muestras que se agrupan en función de un número alto de características comunes). La detección basada en firmas se puede realizar directamente sobre el binario situado en el sistema de ficheros del sistema operativo o bien sobre el binario una vez ha sido cargado y puesto en ejecución por el operativo [Roy06]. En el primer caso, existen numerosos problemas actualmente [Mos07] debido al creciente número de formas de camuflar la muestra original: *packers*, *cryptors*, polimorfismo, etc [CJ04]. En el segundo caso, si bien se solucionan parte de los impedimentos anteriores, dado que se está accediendo directamente al código que se ejecutará, se corre el riesgo de permitir la infección antes de que se reconozca positivamente la muestra. Así mismo, existe una tercera versión para la detección estática que trata de llegar a la ejecución real en el operativo utilizando técnicas de instrumentalización, es decir, tratar de simular la carga y ejecución de la muestra en un entorno suplantado de forma que no se permita la infección y se pueda acceder al contenido real de la misma para realizar la detección [WHF07].

      ii. Detección dinámica o basada en el comportamiento. Otra gran vertiente se basa en la ejecución de la muestra en estudio, bien dejando libertad dentro del sistema operativo, bien emulando el comportamiento del mismo, pero con la

salvedad de que no se conoce previamente la muestra en cuestión. Se realiza un estudio minucioso del malware a reconocer y se extrapolan las características comunes del mismo realizando la captura de evidencias midiendo los diferentes eventos que se produzcan. Una vez seleccionadas las características más relevantes y generalizadoras para el reconocimiento de la muestra se puede realizar la detección de la cualquier muestra que cumpla los patrones establecidos [PCJD08][YSE+07][Dea01][Mil02][Yer06].

(b) Entorno de Red o *Network*:

i. Detección estática. Se compara el tráfico de red con firmas conocidas de ataques y en el caso de coincidencia se genera una alerta. El mantenimiento es bajo debido a que sólo requieren la actualización de una base de datos de conocimiento. La pega de este tipo de sistema radica en que no son efectivos contra ataques nuevos o variaciones de los ataques conocidos. Así mismo, teniendo en cuenta el DPI[1] que se utilice existen diferentes alternativas: basados en autómatas (DFAs[2] y NFAs [3]), basados en sistemas heurísticos y basados en filtros especializados. Los primeros suelen ser implementados dentro de FPGAs[4]y por lo tanto son rápidos al estar directamente en el hardware. Los NIDS basados en sistemas heurísticos tienen la ventaja de no estar tan prefijados como los anteriores, permitiendo variaciones en el *payload*[5]. Finalmente las aproximaciones basadas en el filtrado aportan un uso eficiente de memoria mediante el uso de expresiones regulares para analizar textos, con conjuntos de patrones de longitud variable o con funciones hash secuenciales que aumentan la rapidez de respuesta. Generalmente las soluciones comerciales se encuentra en este apartado.

ii. Detección dinámica. Este tipo de detección pasa por simular

[1] DPI: Inspección Profunda de Paquetes de tráfico de red.

[2] DFA: Deterministic Finite Automaton

[3] Non-Deterministic Finite Automaton

[4] (ing. Field Programmable Gate Array) Circuitos integrados en hardware que soportan programación mediante bloques de lógica implementada sobre funciones de lógica electrónica (AND, OR, XOR, funciones de combinación, puertos de entrada y de salida, conversores ADC/DCA, etc).

[5] *Payload*: Material transmitido a través de paquetes de red.

el tráfico real de una red, dando la impresión de disponer de múltiples equipos y servicios en la misma. Es lo que se denomina como *HoneyPot* y permite el estudio del tráfico que se dirige hacia él de forma genérica (por ejemplo servidores en DMZ) antes de que se dirija el tráfico a su sitio correspondiente. Generalmente no se pasa todo el tráfico hacia este servicio de suplantación y estudio, si no que primero se realiza un filtrado de las evidencias más cercanas al tráfico determinado como sin riesgo por conocimiento experto y redirigiendo sólo aquel tráfico que no cumple la política deseada [Cro03].

2. Basados en Detección de Anomalías

   (a) Entorno Host. La detección de anomalías se desarrolla en dos fases, una de aprendizaje y otra de monitorización/detección. En la primera se trata de aprender el comportamiento considerado como normal. El efecto más destacable de este tipo de aproximación es que se pueden detectar ataques denominados *Zero-Day*[1] no conocidos previamente por el detector. Una limitación de esta técnica de detección es el elevado número de falsos positivos que acarrea y la gran complejidad que existe a la hora de determinar que características deberían ser incluidas en la fase de aprendizaje. Entre las técnicas más destacables en este ámbito destacan: frecuencias de distribución de datos con medición de desviación en la distancia Mahalanobis sobre un centroide [Sto04]; aplicación de técnicas de minería de datos, reglas de asociación y frecuencias de aparición [LS98]; análisis de secuencias cortas de *System-Calls* [Lev97, SHS98][2], estudio de las frecuencias de apa-

[1] Zero-Day Attack. Corresponde a aquel tipo de Malware sobre el que todavía no se tiene información para rellenar una base de datos con su firma o evidencias para detección mediante sistemas heurísticos. Generalmente explotan una vulnerabilidad que o todavía no ha sido descubierta públicamente o todavía no se dispone un parche de seguridad apropiado para la misma.

[2] System-Call. Generalmente un sistema operativo consta de dos partes, la zona kernel o privilegiada y la zona de usuario. La primera realiza las tareas más primarías como acceso a disco, sincronizaciones de objetos, acceso a los componentes hardware, etc y dispone del nivel de privilegio más elevado. La segunda es donde realmente se ejecutan las aplicaciones del usuario. Ambas interactúan entre sí por medio de varios mecanismos. Uno de ellos son las System-Calls, que deberían entenderse como una llamada privilegiada a la zona kernel del sistema operativo.

rición de *System-Calls* [ISG02], análisis de los eventos asociados a un proceso en ejecución [Deb00], grafos de llamadas producidas a la API en zona de usuario [JBT01] etc.

(b) Entorno Network. Suelen ser más complejos a la hora de interpretar el grado de normalidad y requiere de un largo periodo de inicialización o entrenamiento. Así mismo este tipo de detección suele tener un gran número de falsos positivos. Entre este tipo de sistemas se pueden encontrar las diferentes aproximaciones: Análisis estadístico, Redes Neuronales, Agentes Inteligentes Autónomos, Análisis de *Wavelets*, etc. Los trabajos más representativos en el campo engloban: monitorización del número de bytes transmitidos para cada paquete de red (a mayor número para el mismo servicio mayor posibilidad de estar ante un *exploit*) [TAF01] y la anexión en el modelo de anomalías de entradas estáticas [RSZ02].

La comunidad científica ha venido prestando una atención especial a lo que se denominan Sistemas de Correlación. Este tipo de sistemas se caracteriza por aunar diferentes tipos de sensores que permitan la detección más efectiva posible de las amenazas, tanto conocidas como desconocidas, *Zero-Day*. Los sistemas que tienen sensores de ambos tipos generalmente tienen una arquitectura distribuida, en la que como mínimo (dado que suele ser escalable) se presenta un máster y una serie de sondas que recopilan información. Este tipo de sistemas se denomina HbIDS o HyIDS [1] y en ellos la capa denominada como *Low Level* se ha visto complementada con información física del propio hardware o de las instalaciones: alarmas perimetrales, sistemas de control de acceso y un largo etcétera. Igualmente se han unido evidencias procesadas de más alto nivel a este tipo de entornos de monitorización de seguridad. Los sensores que se suelen utilizar son los siguientes:

- Fuentes Low Level:
  - Red: Routers: Cisco, Linksys, Juniper, etc.; Firewalls: Netfilter (IPTABLES/EBTABLES), NuFW, Checkpoint, etc.
  - Host: Logs de sistema, Usuarios, Perfiles de usuario en base a comportamiento (*User Behavior*), Aplicaciones en ejecución, Mo-

[1] *Hybrid IDS*

nitorización de servicios de Red, Trazas de kernel, Auditoría de Procesos, etc.

- High-Level Sources:
  - Honepots: Nepenthes,...
  - Network: Snort, NuFw, Sancp,...
  - Host: HP SIEM, OSSIM, SELinux, Linux PAM, Samhain, Ossec, Prelude, LML, ClamAV,...
  - Scanners: Nessus, p0f, nmap...

Uniendo todos los frentes se ha intentado aportar una solución innovadora que permita gestionar el riesgo de la seguridad de forma integral. En este punto conviene definir el concepto Sistema de Seguridad Perimetral:

Definición 1.1 ***Seguridad Perimetral:***
«Es un concepto emergente que asume la integración de elementos y sistemas, tanto electrónicos como mecánicos, para la protección de perímetros físicos, detección de tentativas de intrusión y/o disuasión de intrusos en instalaciones especialmente sensibles.»

Se ha acunado el concepto de MetaIDS (MIDS) o HbIDS como aquel elemento que se ocupa de mezclar cualquier elemento que pueda mandar información útil para detectar una intrusión, tomando el entorno al completo como un todo. Esta capa de gestión se ocupa de garantizar que los problemas tradicionales de cada una de las piezas integrantes sean menos significativos, bajando el número de falsos positivos/negativos de las mismas.

Pese a que el concepto es innovador y las tendencias en Seguridad Integral siguen este camino, las aproximaciones se han ido realizando en las líneas tradicionales de forma que al final los patrones reconocibles por el MIDS en muchos casos se basan en patrones conocidos de antemano[Tri08]. Otras alternativas a la línea de trabajo anterior, más centrada en la propia arquitectura y la gestión de medidas oportunas para notificación y visualización de los eventos finales, han ido en base a la aplicación de test estadísticos, como por ejemplo Test de Causalidad de Granger, o incluso otros más genéricos llegando a la barrera de la dificultad de establecer los

parámetros correctos para la detección debido a la cantidad de información recabada[MZ07]. Siguiendo la línea de trabajo existen otras aproximaciones que hacen uso de algoritmos particularmente desarrollados para acometer la tesitura de análisis de evidencias y correlación de alertas [ZHR+07].

Igual que en las líneas de trabajo previas, en las que se realizaban estudios sobre entornos simulados denominados *honeypots*, esta nueva vertiente de securización de entornos a nivel integral ha generado nuevos entornos que hacen uso de esta tecnología en busca de la protección. En ellas se dirige el tráfico a una red de estudio en la que se despliega el sistema de correlación de alertas, impidiendo la salida de la misma hasta la verificación intermedia en el entorno suplantado. Esta aproximación permite establecer de antemano el modelo que se podrá aplicar luego internamente [Wu08].

Una posible solución más reciente ha aplicado técnicas de Inteligencia Artificial más punteras a la resolución del problema: Support Vector Machines (SVMs) y K-Means [Ged10]. En ella se vuelve al concepto de detección de anomalías en red y se obtienen resultados satisfactorios, pero se tiene una penalización importante en el cómputo de la detección. Este planteamiento abre una veta poco explorada, realiza de partida una clusterización de los eventos de red, de forma que trata de minimizar el tiempo de cómputo del algoritmo siguiente. Esta aplicación de algoritmos en cascada supone una base para la presente propuesta.

1.2 Retos a afrontar

Se localizan varios puntos de trabajo para problemas que actualmente no tienen una solución única y/o totalmente viable en los que se puede aportar valor a la comunidad (en orden de importancia):

1.2.1 Unificación de modelos de detección de intrusiones

Un objetivo fundamental que se plantea dentro de la Detección de Intrusiones es el de la unificación de las múltiples soluciones de análisis que existen actualmente, fundamentalmente mediante la homogeneización de los diferentes modelos de representación de conocimiento existentes, tanto de detección basada en firmas (usos indebidos) como de detección de anomalías. Para ello, el factor principal que debe tenerse en consideración es la taxonomía de los parámetros de análisis, ya que a partir de ellos y de su morfología deben construirse posteriormente los motores de análisis responsables de la toma de decisiones. De esta forma, en caso de conseguirse un modelo definitivo de representación unificado se pondrían tratar a la vez en un mismo modelo eventos de equipo y de red, basados en conocimiento experto y con el aporte de la detección de anomalías, cubriendo por completo todos los vectores de ataque en la actualidad.

1.2.2 Procesamiento de grandes volúmenes de datos en tiempo real: Big Data.

El primer reto al que se enfrenta la comunidad es la forma en la que abordar una cantidad inmensa de datos. Es un problema de raíz a la hora de manejar los datos y poder llegar a conclusiones acertadas el poder procesar toda la información sin sintetizarla u obviar parte de la misma. Además, a la problemática planteada de la cantidad de datos, se hace necesario tener una capacidad de reacción en tiempo real para impedir la infección de un sistema operativo. En el momento en que se descubre una vulnerabilidad y se comienza un ataque basado en ella, se carece de información experta para realizar la detección con lo que, la única base con la que se cuenta es la información proporcionada de base por las sondas del HbIDS, que es tan extensa que para llevar a cabo técnicas algorítmicas habría que destinar unos recursos de hardware altísimos para realizar el análisis al momento. Otra pega asociada es que si para un mismo evento se genera demasiada in-

formación o información repetida puede llevar al sistema a sufrir un DoS[1]. Esto abre la puerta al mundo de las tecnologías Big Data, donde el escalado del procesamiento trata de ser lineal en cuanto al número de máquinas dedicadas al problema.

1.2.3 **Dificultad en la detección de ataques noveles (Zero-Day)**

Los ataques no conocidos representan un problema grave a nivel mundial como se introducido (Figuras 1.2 y 1.4). En este caso los sistemas basados en detección de anomalías son identificados como la solución, si bien suelen conllevar un exceso de falsos positivos/negativos según se alejan del modelado que se realiza en los mismos. Así mismo, la sección de variables adecuadas que permitan dar visión del problema es un paso previo para poder medir la anomalidad. También puede ocurrir que, aun teniendo las variables a medir, si exclusivamente se ha basado la detección en conocimiento experto seguramente no estará contemplada la nueva vulnerabilidad, pese a las firmas generalistas que se utilizan en los grandes fabricantes. Ocurre lo mismo con aquellos sistemas que van un paso más allá de las firmas conocidas o del conocimiento experto representado en forma de reglas de comportamiento (intercepción de bajo nivel de host o de red), porque si se ha propiciado un aprendizaje en la Inteligencia Artificial que se ocupa de la detección sobre ese mismo conocimiento, se detectarán pequeñas variaciones del mismo pero no una brecha completamente nueva.

1.2.4 **Dificultad de procesamiento de fuentes heterogéneas.**

Sumado al apartado anterior, la comunidad se enfrenta en este punto a la dificultad de seleccionar aquellas variables representativas para cada tipo de ataque. Este hecho hace que muchos proyectos o líneas de trabajo planteadas se ciñan exclusivamente a estudiar ciertos tipos de ataques dejando de lado al resto de ataques conocidos. Si además se añade el hecho de que sin una selección certera de variables se va a despreciar ciclos de cómputo en analizar datos no relevantes, se demuestra claramente que es necesario un mecanismo para selección de las variables representativas para cada entorno. Así mismo, se hace necesario la creación de modelos integrales que

[1] DoS: Denial Of Service. Ataque a un sistema por el que se queda sin recursos para seguir dando servicio.

aúnen tanto variables de entorno *Host* con variables de entorno *Network*, sobre una arquitectura escalable que haga uso del potencial de las tecnologías de *Cloud Computing* actuales. Finalmente, el punto más importante es el integrar o unificar informaciones provenientes de fuentes heterogéneas, que envían la información sin seguir un estándar o que no tiene forma de anexionarse de forma sencilla, requiriendo complejos procesamientos antes de poder operar con los datos en su conjunto.

1.2.5 **Alta tasa de falsos positivos/negativos en las detecciones de los sistemas basados en anomalías.**

Una solución a los ataques noveles, como se ha introducido en los puntos anteriores, viene siendo el uso de sistemas basados en anomalías. Estos sistemas tienen la ventaja de detectar cualquier cambio fuera de lo normal, con la pega inherente de que contra más grande es el entorno de estudio, mayor número de personas están involucradas y con ello la aleatoriedad de los eventos puede llevar a los sistemas de detección a error. Se debe tener en cuenta que generalmente este tipo de detectores no tiene un aprendizaje continuo, con lo que cualquier cosa no contemplada previamente supone una falsa alarma. Si el MIDS además tiene acceso a medidas de control preventivas, como por ejemplo matar procesos y/o cortar comunicaciones, una falsa supone la pérdida de eficiencia y una falta de disponibilidad de servicios o sistemas que pueden ser críticos. Si en cambio el MIDS sólo ofrece una visualización para poder gestionar la seguridad hará que los administradores del mismo dejen de utilizarlo por la desconfianza que pueda generar o incluso la saturación de notificaciones que pueda entregar.

1.2.6 **Dificultad al prescindir de conocimiento experto.**

Los sistemas de este tipo deben tener representado en su interior el conocimiento experto en la materia, bien sea combinando con otras técnicas de Inteligencia Artificial, bien sea en conjunción con un sistema de detección de anomalías. Pero es necesario dotar al sistema del conocimiento experto para poder dirigir el aprendizaje del mismo en la dirección apropiada. Si por ejemplo se plantea un sistema sin este conocimiento en su interior y se le deja libertad de aprendizaje no-supervisado, cabe la posibilidad de efectuar un ataque dirigido de forma que poco a poco un atacante pueda conducir al sistema hasta el punto que no detecte (técnicas de *eavesdropping*). El conocimiento experto es crucial para los sistemas de la Seguri-

dad de la Información, pero debe ser compensada con un grado de libertad apropiado para que ese conocimiento/experiencia sea extrapolable a otras situaciones parecidas sin requerir la intervención humana continuamente en la generación de reglas.

1.3 Hipótesis y Objetivos

1.3.1 Axiomas propuestos de corte genérico

A continuación se va a proceder a establecer una serie de hipótesis de corte general a la presente propuesta:

1. *Todo malware sólo representa un problema en la seguridad cuando se pasa a ejecución.* Es indiferente que la muestra en cuestión se encuentre almacenada en disco, con lo que no se estima necesario el reconocimiento de la misma en el sistema de ficheros.

2. *Un ataque de red sólo representa un problema en la seguridad cuando existe un servicio de red que se vea afectado.* Pese a ser ataques conocidos, no se debe dejar conducir el aprendizaje de los sistemas expertos basados en inteligencia artificial sobre aquellos datagramas que no representen una brecha real para los sistemas a proteger.

3. *La seguridad integral de un entorno correlaciona directamente los eventos de red con los eventos producidos en los hosts de la misma.* Si un malware va a proliferar en un entorno corporativo hará uso de la red de forma anómala. Un ataque a una red sólo es perjudicial si el tráfico anómalo es recibido y tramitado por un sistema operativo de Host o de *Appliances* de Red (windows, linux, BSD, novel, cisco ios, etc).

1.3.2 Hipótesis

En función de lo explicado en los puntos anteriores se va a formular la hipótesis central de la propuesta de tesis:

> *«Existe un modelo, que trabajando en tiempo real, permite correlacionar evidencias producidas dentro del sistema operativo y de la red simultáneamente para detectar ataques a la Seguridad la Información. Este modelo, integraría tanto la Detección de Usos Indebidos como la Detección de Anomalías, incluyendo para ambos casos el Conocimiento Experto dentro de una capa de Inteligencia Artificial.»*

Se formula la anterior hipótesis para demostrar la viabilidad del estudio e implementación de un modelo basado en Inteligencia Artificial que pueda ser la base para la implementación de nuevos sistemas de detección de intrusiones a nivel integral: *Host + Network*. Permitiendo la reacción temprana y el análisis en tiempo real de los eventos acaecidos, la presente tesis se fundamenta en la hipótesis de que es posible establecer un modelo de protección basado en evidencias expertas que, extrapoladas automáticamente, correlacionan eventos de *Network* con eventos de *Host* para securizar un determinado entorno informático. Este modelo basado en capas en cascada de algoritmos de Inteligencia Artificial se generará para el entorno en cuestión mediante estudio de diferentes métricas y podrá ser implementado y desplegado de forma confiable dentro de ese entorno garantizando la seguridad del mismo, apoyándose en las tecnologías de las granjas de supercomputación más recientes. Las técnicas para integrar el conocimiento experto se categorizan dentro del aprendizaje supervisado como *Clasificación*, mientras que aquellas técnicas de aprendizaje no-supervisado denominadas *Clusterización* realizarán la detección de anomalías. El modelo integral que plantea la tesis, aparte de reaccionar en tiempo real, asevera que es posible unir homogéneamente ambas fuentes heterogéneas para obtener un resultado que permita la toma de decisiones posterior.

1.3.3 **Objetivos**

En base a los axiomas propuestos y la hipótesis fundamental se establece el siguiente objetivo principal:

> *«Creación de un nuevo modelo Big Data para el área de la Seguridad de la Información, que se integre con las soluciones actuales, y que permita aplicar técnicas de Inteligencia Artificial en Cascada correlacionando evidencias de fuentes heterogéneas (Host, Red, etc.), con respuesta en tiempo quasi real y que tenga escalabilidad en base al número de máquinas usadas en el cluster.»*

Para la consecución del objetivo principal se hace necesario la realización de una serie de objetivos específicos que pasarán a ser detallados en los siguientes puntos:

1. Definición de la arquitectura más conveniente para abordar el objetivo principal.

2. Selección de las tecnologías de Big Data que mejor puedan alojar al modelo planteado en el objetivo principal.

3. Búsqueda y generación de evidencias adecuadas para validar el modelo en cuestión, teniendo en cuenta tráfico normal y ataques dirigidos.

4. Selección, despliegue y desarrollo de los sistemas que permitan la recolección de evidencias de *host*.

5. Selección, despliegue y desarrollo de los sistemas que permitan la recolección de evidencias de red.

6. Establecimiento de una metodología para aplicación de las tecnologías Big Data al procesamiento de la información anterior, permitiendo identificar las variables más representativas de la capa de adquisición.

7. Localizar una combinación de algoritmos que con una tasa de falsos positivos/negativos baja aceptable que pueda ser aplicado en el campo de la Seguridad de la Información.

Así mismo se identifican los siguientes objetivos operacionales a la hora de conseguir el objetivo final:

1. Diseñar e implementar un mecanismo de captura de evidencias de bajo nivel del sistema operativo.

2. Diseñar e implementar un mecanismo de captura de evidencias de bajo nivel en el entorno de red.

3. Creación de un modelo de datos apropiado para poder recoger la información de los mecanismos anteriores: la capa de adquisición.

4. Diseñar e implementar un sistema capaz de recorrer la información recabada en tiempo real, permitiendo el acceso a los datos recogidos.

5. Estudio teórico de los diferentes algoritmos que se puede aplicar sobre el modelo de datos anterior buscando el tiempo óptimo de procesamiento por cada nivel del clúster.

6. Diseño e implementación de los algoritmos seleccionados para crear una clusterización jerárquica por niveles capaz de acometer la detección de anomalías.

1.4 Metodología

El objetivo final de esta tesis doctoral es la consecución de unos conocimientos avanzados en el área de estudio, así como la realización de un aporte beneficioso a la comunidad de investigación en el área de Seguridad de la Información. Para conseguir los objetivos planteados en el apartado anterior se realizarán las siguientes fases, acorde al modelo de investigación denominado *Sand-Clock* [TD01, Der14], que conducen la investigación desde la generalidad: identificación del área de trabajo, revisión bibliográfica; hasta lo más específico: planteamiento de las preguntas de investigación, experimentación, observación y análisis e interpretación de resultados; volviendo a la generalidad para aplicar: la discusión con la comunidad científica, las conclusiones de la investigación y el trabajo a futuro. El proceso seguido debe ser cíclico y trabajando con las conclusiones obtenidas en la mejora de los métodos propuestos llegando incluso a reformular la hipótesis caso de ser necesario. La metodología seguida se resume de la siguiente forma:

1. Estudio área de trabajo en Seguridad de la Información e identificación de retos. Para poder afianzar los conceptos y avanzar en el área señalada se deberá realizar un estudio exhaustivo de los últimos avances, estudiando los puntos abiertos y tratando de complementarlos con otras áreas de estudio para ver si alguna puede complementar al área principal de Seguridad de la Información y sus retos actuales.

2. Revisión bibliográfica. Una vez delimitada el área en estudio y sus retos más acuciantes, llega el momento de revisar las publicaciones científicas más relevantes y recientes, ahondando en la historia de cada tema para ver líneas de trabajo sin salida o no factibles por limitaciones técnicas. Este estudio culminará en la realización del estado del arte mostrando las tendencias más actuales de la comunidad y los retos específicos a resolver, permitiendo centrar la investigación subsiguiente.

3. Planteamiento de hipótesis, objetivos y plan de trabajo. Se deberá delimitar la hipótesis de trabajo, en una o varias, definiendo objetivos que guiarán el trabajo de investigación que permitan validar la hipótesis planteada. Así mismo, sobre esos objetivos se deberá realizar un plan de trabajo plausible, con plazos e hitos.

4. Desarrollo de soluciones técnicas apropiadas a cada objetivo. Como

se ha planteado en el apartado anterior, habrá que realizar el diseño y el desarrollo de ciertas aplicaciones en diferentes estadios del proceso. Algunas de estas tareas conllevarán el estudio, aprendizaje y las pruebas de técnicas pertenecientes a otros campos de investigación. En concreto para esta investigación: Bases de Datos Distribuidas, Bases de Datos no Relacionales, Sistemas Distribuidos de procesamiento de datos basados en flujos, Sistemas Distribuídos para Supercomputación de Algoritmos de Inteligencia Artificial, bajo nivel de sistemas operativos y de redes para recolección de evidencias, y un largo etcétera.

5. Desarrollo e implementación de nuevas metodologías. Se realizará el diseño y la implementación que dé cabida al objetivo principal planteado en la sección 1.3.3.

6. Evaluación de las soluciones planteadas. Se irán realizando pruebas para medir el aporte de valor de cada una de las lineas que se empleen en el proceso, por lo que la presente tesis puede servir para descartar ciertas metodologías como objetivo secundario.

7. Interacción con la comunidad científica. Se estará al tanto de las novedades en el área y si es posible se establecerán vínculos con otras fuentes del entorno de forma que la relación pueda ser positiva para ambos extremos.

8. Difusión de los resultados. Una vez dados por concluidos los objetivos planteados se pasará a la comunicación pública de los mismos, bien foros especializados, bien en base a conferencias, bien en base a publicaciones en el área.

1.5 Estructura del documento

En total hay 8 capítulos, distribuídos en dos partes: Parte I: El propio framework del modelo planteado, y la Parte II: Experimentación y análisis de resultados.

El capítulo 1 presenta el contexto general y las principales razones que motivan la pertinencia de la investigación. Plantea los retos, la hipótesis fundamental, los objetivos, la metodología que guía la investigación y la relevancia.

En el capítulo 2 se realiza el estado del arte comenzando por (i) los conceptos fundamentales del área de la Seguridad de la Información, (ii) repasando las técnicas aplicadas a los Sistemas de Detección de Intrusiones de red (NIDS), (iii) las técnicas empleadas en los Sistemas de Detección de Intrusiones de host (HIDS) y (iv) finalmente se centra en la revisión de la aplicación de conceptos de Big Data en Sistemas de Detección basados en Fuentes Heterogéneas.

La parte I, incluye los capítulos 3 y 4. El capítulo 3 presenta los métodos empleados para la Recogida de Evidencias de fuentes Heterogéneas, mientras que el capítulo 4 describe la tecnología empleada en Big Data y las arquitecturas de aplicación en los sistemas de detección/prevención de intrusiones del modelo planteado en la tesis (ESIDE-BIDS).

La segunda parte, Parte II, está dedicada a la experimentación, análisis de resultados y validación del modelo planteado. En concreto, el capítulo 5, aplica técnicas de machine learning (aprendizaje supervisado y no supervisado) sobre el *dataset* KDDCup99. El capítulo 6, se ocupa de la recogida de evidencias, procesamiento y escalabilidad del modelo para análisis en tiempo real de la información. Y el capítulo 7, donde se revisan los diferentes modelos de machine learning susceptibles de integración en tiempo real como motores del modelo planteado en la tesis. Para finalizar, el capítulo 8, presenta las principales contribuciones, conclusiones de la tesis, las limitaciones de la investigación y las líneas de trabajo futuras para investigaciones posteriores.

Capítulo

2

Estado del Arte

Actualmente en prácticamente todas las organizaciones con presencia en Internet, o que disponen de un servicio de Tecnologías de la Información (CAU) para mantenimiento de los PCs y Servidores de la misma, suelen disponer de soluciones para Detección/Prevención de Intrusiones. Estas soluciones pueden ser de varios tipos, bien comerciales, bien basadas en software libre, aunque el grueso de las mismas se centra en la Detección de Usos Indebidos, es decir, en reglas con mayor o menor nivel de generalización.

La historia de este tipo de tecnología, los Sistemas de Detección de Intrusiones, se remota a la década de los ochenta. En 1980, las Fuerzas Armadas estadounidenses encargaron a James P. Anderson un estudio sobre la clasificación automática de los eventos que se registraban en los sistemas de auditoría de los Sistemas Operativos. La clasificación debía servir para detectar actividades subversivas, basándose en el comportamiento habitual de los usuarios [And80].

Posteriormente, a mediados de los años 80, el Gobierno Estadounidense solicitó al SRI Internacional nuevos trabajos de investigación para analizar los registros de auditoría de los sistemas informáticos gubernamentales. Phd. Dorothy Denning creó un primer modelo general de un Sistema de Detección de Intrusiones [Den87], que lanzó posteriormente el proyecto IDES [JV91, LT90]. Las conclusiones de estas investigaciones han sido el

referente internacional más importante para el área de los Sistemas de Detección de Intrusiones, y más concretamente para el campo de Detección de Anomalías dentro del área de trabajo anterior

Paralelamente a la investigación del proyecto IDES, el Laboratorio Lawrence Livermore de la Universidad de California abordó el desarrollo del proyecto Haystaceyk, cuya meta era la construcción de una nueva versión del Sistema de Detección de Intrusiones para las Fuerzas Aéreas que basaba la detección mediante la comparación de los registros de auditoría con patrones de ataque predefinidos [S.88]. Este proyecto dio origen a lo que se conoce como Detección de Usos Indebidos. Basándose en el proyecto anterior, se desarrollo un nuevo proyecto denominado DIDS [SS91], que formaliza la primera versión de los sistemas de Detección de Intrusiones Distribuidos.

Más adelante, a principio de los 90, la Universidad de California presentó la idea de la Detección de Intrusiones orientada a la Red, lanzando el proyecto NSM y revolucionando el sector para la época [HL90]. El equipo que estaba trabajando en el proyecto DIDS recogió las ideas aportadas sobre los sistemas basados en red, creando por primera vez el concepto de Sistemas Híbridos (HbIDS ó HyIDS).

En este ámbito, durante las siguientes décadas, la tecnología ha ido madurando, convergiendo en las principales tendencias y pasando de proyectos de investigación (NIDES [AFTV94], EMERALD [yNP97], STAT [VG00, IK95] a tecnologías comerciales (ISS de RealSecure, SNORT, SURICATA, OSSIM, OSSEC, etc.), hasta el punto de que hoy en día se ofrecen productos integrales denominados SIEM proporcionados por grandes fabricantes de software-hardware (IBM, HP, etc.) o sistemas de antivirus (Symantec, McAfee, etc.). Si bien se debe indicar que la inmensa mayoría de las soluciones existentes se siguen basando en la Detección por Usos Indebidos.

De todo lo anterior se hará un estudio en profundidad en las siguientes secciones.

2.1 Conceptos Fundamentales en el ámbito de Seguridad de la Información

Es importante presentar algunos conceptos básicos, sobre los cuales está basada la presente tesis. En primer lugar se definen algunos conceptos fundamentales de Seguridad de la Información, y a continuación se pasa a definir conceptos básicos de Detección de Intrusiones.

2.1.1 Conceptos Generales

Definición 2.1 ***Seguridad:***

"(1) Measures taken to protect a system. (2) The condition of a system that results from the establishment and maintenance of measures to protect the system. (3) The condition of system resources being free from unauthorized access and from unauthorized or accidental change, destruction, or loss."

Definición 2.2 ***Seguridad de la Información:***

"The protection of data from disclosure, alteration, destruction, or loss that either is accidental or is intentional but unauthorized." [R.00]

"The condition of a system that results from the establishment and maintenance of measures that assure information confidentiality and integrity, availability, authenticity and non-repudiation in computer and communication systems." [NSA06, G.02]

Definición 2.3 ***Confidencialidad:***

"The property that information is not made available or disclosed to unauthorized individuals, entities, or processes; i.e., to any unauthorized system entity." [Ins05, R.00, IM03]

Definición 2.4 ***Integridad:***

"The property that data has not been changed, destroyed, or lost in an unauthorized or accidental manner." [Ins05, R.00, IM03]

Definición 2.5 ***Disponibilidad:***

"The property of a system or a system resource being accessible and usable upon demand by an authorized system entity, according to performance specifications for the system; i.e., a system is available if it provides services according to the system design whenever users request them." [Ins05, R.00, IM03]

Definición 2.6 ***Autenticidad:***

"The property of being genuine and able to be verified and be trusted." [Ins05, R.00, IM03]

Definición 2.7 ***Repudio:***

"Denial by a system entity that was involved in an association (especially an association that transfers information) of having participated in the relationship." [Ins05, R.00, IM03]

2.1.2 Conceptos de Detección de Intrusiones

A continuación se describen los términos fundamentales de Detección de Intrusiones que se usarán frecuentemente a lo largo de esta tesis:

Definición 2.8 ***Ataque:***

"An assault on system security that derives from an intelligent threat, i.e., an intelligent act that is a deliberate attempt (especially in the sense of a method or technique) to evade security services and violate the security policy of a system." [R.00]

Definición 2.9 ***Intruso:***

"An entity that gains or attempts to gain access to a system or system resource without having authorization to do so." [R.00]

Definición 2.10 ***Intrusión:***

"A security event, or a combination of multiple security events, that constitutes a security incident in which an intruder gains, or attempts to gain, access to a system (or system resource) without having authorization to do so." [R.00]
"Any set of actions that attempt to compromise the integrity, confidentiality, or availability of a computer resource." [D.01]

Definición 2.11 ***Detección de Intrusiones:***

"A security service that monitors and analyzes system events for the purpose of finding, and providing real-time or near real-time warning of, attempts to access system resources in an unauthorized manner. " [R.00]
'Intrusion detection is the process of monitoring the events occurring in a computer system or network and analyzing them for signs of intrusions, defined as attempts to compromise the confidentiality, integrity, availability, or to bypass the security mechanisms of a computer or network. Intrusions are caused by attackers accessing the systems from the Internet, authorized users of the systems who attempt to gain additional privileges for which they are not authorized, and authorized users who misuse the privileges given them. Intrusion Detection Systems (IDSs) are software or hardware products that automate this monitoring and analysis process." [BM01]
"Security management system for computers and networks. An Intrusion Detection System gathers and analyzes information from various areas within a computer or a network to identify possible security breaches, which include both intrusions (attacks from outside the organization) and misuse (attacks from within the organization)." [Ins05]

Definición 2.12 ***Sistema de Detección de Intrusiones orientado a la Máquina (Host Intrusion Detection System, HIDS):***

"Intrusion detection systems that use information from the operating system audit records to watch all operations occurring on the host that the intrusion detection software has been installed upon." [Ins05]

Definición 2.13 ***Sistema de Detección de Intrusiones orientado a la Red (Network Intrusion Detection System, NIDS):***

"Intrusion Detection System that monitors the traffic on its network segment as a data source." [Ins05]

Definición 2.14 ***Detección de Usos Indebidos (Misuse Detection):***

"Detection method that analyzes system activity, looking for events or sets of events that match a predefined pattern of events that describe a known attack." [BM01]

Definición 2.15 ***Sistema de Detección de Anomalías:***

"Analysis method that identifies any unacceptable deviation from expected behavior. Expected behavior is defined, in advance, by a manually developed profileor by an automatically developed profile. An automatically developed profile is created by software that collects and processes characteristics of system behavior over time and forms a statistically valid sample of such behavior. Note that unexpected behavior is not necessarily an attack; it may represent new, legitimate behavior that needs to be added to the category of expected behavior." [ACFW00]
"Detection method that identifies abnormal unusual behavior (anomalies) on a host or network." [BM01]

Definición 2.16 ***Falso Positivo (False Positive):***
"An event, incorrectly identified by the Intrusion Detection System as being an intrusion when none has occurred." [yNT98]

Definición 2.17 ***Falso Negativo (False Negative):***
"An event that the Intrusion Detection System fails to identify as an intrusion when one has in fact occurred." [yNT98]

Definición 2.18 ***Sistema de Prevención de Intrusiones (Intrusion Prevention System):***
"System that actively monitor a network or host for attacks and block those attacks from occurring." [Fon03]

2.1.3 Taxonomía de los Sistemas de Detección de Intrusos

El primer modelo de Sistemas de Detección de intrusiones fue desarrollado por Dorothy E. Denning en 1987 [Den87], y aunque es un modelo simplificado se ajusta bien a una arquitectura genérica [LKS05] , que tiene los siguientes componentes, Figura 2.1:

1. Detector: Se ocupa de procesar los datos recogidos de los sensores para clasificar las actividades intrusivas.

2. Sensores: Se ocupan de recoger la actividad del sistema monitorizado.

3. Modelo de conocimiento: Se puede entender como una base de datos que contiene tanto la información envíada por los sensores, como las pautas para realizar la detección.

4. Configuración: Es el componente que proporciona información acerca del estado que tiene el sistema de Detección de Intrusiones.

5. Componente de Respuesta: Se ocupa de iniciar las acciones necesarias cuando se ha detectado una intrusión, pudiendo ser: activa, cuando la acción es automatizada; pasiva, cuando involucra intervención humana.

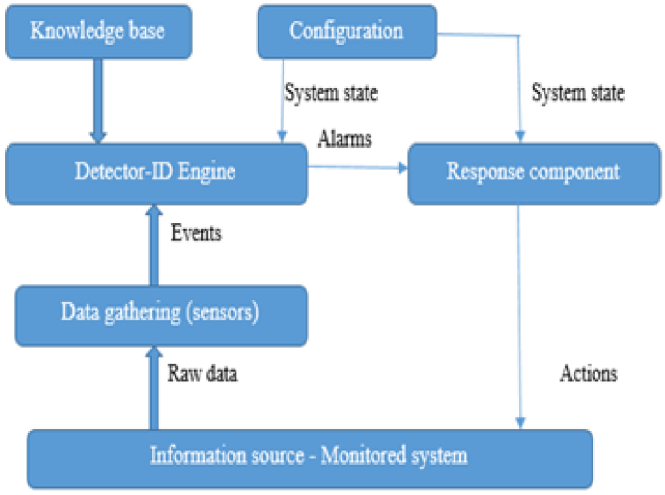


Figura 2.1: Modelo de IDS propugnado por Denning.

La estandarización de los Sistemas de Detección de Intrusiones dispone de dos estándares internacionales. El primero viene fomentado por *America Defense Advanced Research Projects Agency* (DARPA) y se denomina *Common Intrusion Detection Framework* (CIDF). El segundo es propuesto por *Internet Engineering Task Force* (IETF) bajo el nombre de *Intrusion Detection Working Group* (IDWG). Las características de cada uno de ellos son las siguientes:

1. CIDF. Consiste principalmente en cuatro partes: la estructura del IDS, el sistema de comunicaciones, el lenguaje de descripción y la API. Así mismo, CIDF plantea cuatro tipos de componentes básicos: generadores de eventos, analizadores de eventos, unidades de respuesta y la base de datos. CIDF recoge información de red, así como de sistema operativo, en forma de logs y/o trazas de auditoría.

2. IDWG. El grupo de trabajo ha creado un estándar de comunicación entre las partes implicadas denominado IDMEF (*Intrusion Detection*

Message Exchange), entendido como una definición XML Object-Oriented. También disponde de un protocolo para transmisión de alertas (IAP, *Intrusion Alert Protocol*) que combina cifrado y autenticación para una transmisión adecuada y segura de las mismas.

Realizando un estudio en la literatura existente hasta la fecha sobre la taxonomía de los Sistemas de Detección de Intrusiones (IDS), se pueden clasificar atendiendo a varios criterios [Ran15]:

1. Clasificación basada en la arquitectura:
   (a) Centralizada:
      i. Todas las operaciones se realizan en la misma máquina.
      ii. Es el modelo más simple de realizar.
      iii. Tiene un punto único de fallo.
   (b) Distribuida:
      i. Se divide generalmente mínimamente en dos componentes: sensores y consola central.
      ii. La consola se ocupa de monitorizar eventos, controlando a los sensores.
      iii. El motor central almacena la actividad y genera las alarmas.
      iv. Requiere de unas comunicaciones seguras entre los elementos, así como procesar varios tipos de formato de datos.
2. Clasificación basada en las técnicas de análisis:
   (a) Sin estado:
      i. Los eventos se tratan con independencia del resto.
      ii. Es el más sencillo de diseñar.
      iii. Tiene una velocidad de procesamiento muy alta.
   (b) Con estado:
      i. Mantiene toda la información de los eventos anteriores.
      ii. El efecto de un evento concreto depende de la posición que ocupe en el flujo de eventos previos.
      iii. Tiene un diseño complejo.
      iv. Es más efectivo a la hora de detectar ataques distribuidos.
3. Clasificación basada en las técnicas de detección:

(a) Detección de usos indebidos: El punto más fuerte de esta técnica es para detectar ataques conocidos, ya que reconoce la intrusión buscando patrones conocidos de tráfico y trazas de aplicaciones que se sabe son maliciosos. El punto más significativo de este tipo de motores es la creación de reglas suficientemente generalistas para poder capturar variaciones del ataque en cuestión, evitando *Falsos Positivos*. El punto más crítico es que no se pueden detectar ataques nuevos. Las técnicas más empleadas suelen ser: Sistemas Expertos, Análisis de firmas y Análisis de Transiciones de Estado.

   i. Sistemas basados en Host. Recogen la información de eventos del sistema operativo.

   ii. Sistemas basados en Red. Recogen la información de eventos de red. [1]

(b) Detección de Anomalías: Se identifican los eventos mediante categorización automática como anómalo o normal, por lo que requiere de una fase de aprendizaje o entrenamiento. Este tipo de motores de detección pueden detectar ataques *Zero Day*, o no conocidos, si bien suelen adolecer de grandes tasas de *Falsos Negativos*.

   i. Basados en Host o en Red. Igual que en el caso anterior.

   ii. Basados en Estadística:

      A. Operacionales o basados en límites de métricas. Este tipo de sistemas se usa cuando se puede reconocer un ataque por comparación con una evidencia observable al superar un determinado límite. Ejemplo: *logins* fallidos en un determinado momento fuera de horas de trabajo [LKS05].

      B. Modelos de Markov: Se puede usar esta técnica para tratar de reconocer un determinado estado en base a los anteriores, atendiendo a los eventos que se están recibiendo en este momento. [JPP11]

      C. Estadísticos: Media, varianza, desviación típica Se trata de modelar la normalidad basándose en la observación y ofreciendo un rango de confianza para los parámetros observados [LKS05].

[1] Se ampliará esta definición más adelante con sistemas más complejos que mezclan Host con Red: Hybrid IDS; e incluso Sistemas Collaborative Intrusion Detection Systems.

D. Modelos Multivariable: Se fundamenta en los modelos de desviación estándar anteriores. La diferencia radica en que estos modelos se basan en la correlación entre dos o más métricas [LKS05].

E. Series Temporales: Estos modelos tratan de reconocer las anomalías por revisión del orden y el intervalo temporal de las actividades de la red. Si la probabilidad de que aparezca un determinado evento es baja se considera anormal [LKS05].

iii. Basados en Conocimiento:

A. Transición de Estados Finitos: Este tipo de análisis hace uso de la transición de estados para categorizar las intrusiones. El diagrama de transiciones no es más que un gráfico de las intrusiones como una secuencia de estados variables que llevan desde un estado original de protección a un estado destino de intrusión. Se modelan sólo los eventos dañinos o peligrosos de auditoría del sistema operativo que completan la intrusión (STAT, State Transition Analysis Tool) [PAGT07]. Esta técnica es capaz de modelar ataques complejos, variaciones de ataques conocidos y ataques producidos a través de múltiples cuentas de usuario.

B. Sistemas Expertos: En los Sistemas Expertos, el conocimiento se modela a través de un conjunto de reglas, que consisten en dos partes: antecedente, describiendo cuándo deberán ser aplicadas; y consecuente, que describe las acciones que deberán tomarse cuando se da el antecedente. Una regla determinada se lanza cuando las técnicas de *pattern-matching* determinan que se cumplen los antecedentes de la misma. El mantenimiento y la actualización de un sistema de este tipo suele ser complejo [PAGT07].

C. Scritps Descriptivos: Hay numerosos lenguajes de scripting que pueden describir las firmas de los ataques en ordenadores o redes, proporcionados por la comunidad de expertos en Intrusiones. Todos estos scripts son capaces de identificar las secuencias de eventos particulares que revelan ataques [JPP11].

iv. Basados en Minería de Datos:

A. Clusterización: Es una técnica de aprendizaje no supervisado para encontrar patrones en datos no etiquetados con muchos atributos en cada vector de evidencias. Un ejemplo muy utilizado en el área es K-means [GRY12], Las aproximaciones más actuales en este ámbito: Density-Based Methods, K-mean Clustering [MRR12] , Mode Based Methods, Partitioning Methdos y Hierarchy Methods [HSC+11].

B. Reglas Asociativas: Este tipo de sistemas se usan en la detección de anomalías donde los datos en crudo se transforman en paquetes ASCII, obteniendo la información real de las conexiones establecidas. Sobre estas conexiones se crean modelos que detectan las intrusiones que salen fuera de las reglas aprendidas en las mismas.

C. Clasificación: Se clasifican las evidencias de datos como normales o anómalas. Las aproximaciones actuales vienen de la mano de Árboles de Decisión (DT, ID3, C4.5, C5.0, etc.) [MRR12, PP07, Gol14]. La clasificación generalmente incluye los siguientes pasos: (1) Se identifican las clases y los atributos de cada una de los datos de entrenamiento, (2) se identifican los atributos óptimos que permiten la clasificación de cada clase, (3) se aprende un modelo en base al conjunto de datos de entrenamiento, (4) se usa el modelo aprendido para clasificar datos no conocidos y evaluar el modelo.

v. Basados en técnicas de *Machine Learning*:

A. Redes Bayesianas: Una red bayesiana es una representación que establece las relaciones de probabilidad entre las variables o atributos de un determinado vector de evidencias. Se suele usar en combinación de los sistemas Basados en Técnicas de Estadística, ya que consigue extraer las interdependencias entre las variables y puede cubrir casuísticas donde hay ausencia de datos, así como de descubrir relaciones existentes difíciles de identificar. Se suele trabajar con ellas en forma Naive, cuando hay una alta independencia entre los atributos, o en formato completo cuando las redes son complica-

das. Así mismo, existe una alternativa denominada Dynamic Bayesian Network, que crea el concepto de temporalidad, desplegando una Red Bayesiana en múltiples estados temporales [PP07, SM12].

B. Redes Neuronales: Estas son una de las técnicas de machine learning más utilizadas en el área. Las redes denominadas ANN (*Artificial Neural Networks*), si bien muy usadas, tienen una pega significativa: tienen una precisión de detección muy baja y una estabilidad de detección débil, especialmente para ataques poco frecuentes [WHMH10].

C. Algoritmos Genéticos: En general esta técnica es un tipo de algoritmo de búsqueda que sirve para resolver varias dificultades, creando una solución óptima a un problema. Se comienza con una población de cromosomas, denominada población, que se permitirá mutar con una función hasta que se cumpla una situación predefinida de finalización [Su11].

D. Lógica Difusa: Las técnicas de lógica difusa se usan en el área desde 1990 y generalmente se aplican en combinación con las técnicas de minería de datos, permitiendo la reducción del tamaño de los conjuntos de datos y la selección de características que indican anomalías [PAGT07].

E. Máquinas de Vector-Soporte (SVM): Este método de machine learning se usa en la clasificación, siendo uno de los métodos más robustos y precisos, con la pega del gran consumo de memoria y la complejidad computacional que implica [BG12].

Desde la aparición de la Detección de Intrusiones se ha venido siguiendo un esquema de funcionamiento común, donde las responsabilidades de estos sistemas han sido: la captura de datos del sistema de información, el análisis de dichos datos para tratar de detectar algún tipo de actividad subversiva a partir de ellos, y el proporcionar una respuesta ante aquellas situaciones susceptibles de comprometer la seguridad del sistema. En general, estas tres fases se llevan a cabo de forma secuencial y típicamente ininterrumpida, ya que la mayoría de IDS se plantean el objetivo de responder ante eventos del sistema en tiempo real [MB04, LSC+01].

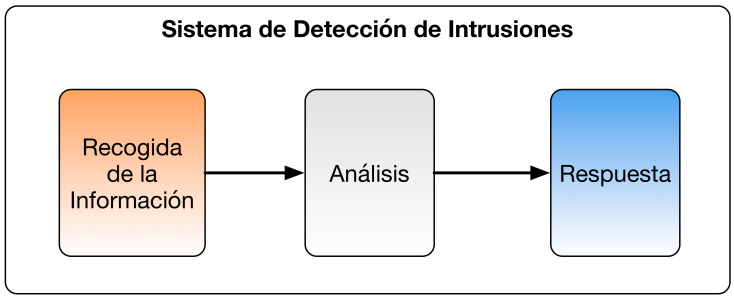


Figura 2.2: Modelo General de Funcionamiento.

2.1.3.1 Recogida de información

Respecto a la etapa la recogida de información, las dos filosofías de funcionamiento que se han introducido anteriormente: Detección orientada a la Máquina (*Host*) y Detección orientada a la Red (*Network*). Sin embargo, ambos métodos muestran importantes debilidades, que hacen necesario un profundo estudio, de cara a la consecución de un modelo híbrido que maximice las capacidades reduciendo los inconvenientes [E.04, Ho01, M.01b, D.01, ACFW00, K.99].

2.1.3.1.1 Detección de Intrusiones orientada a la Máquina.

La Detección de Intrusiones orientada a la Máquina fue el primer enfoque de diseño sobre el que se empezó a trabajar en los orígenes de la Detección de Intrusiones [And80], y constituye probablemente la rama sobre la que tradicionalmente se ha desarrollado un mayor número de investigaciones. En general, este tipo de detección está considerado como el potencialmente más exacto en la toma de decisiones, debido simplemente a su ubicación dentro del propio sistema que se pretende proteger. Este emplazamiento permite que el Sistema de Detección pueda determinar con precisión los usuarios y procesos que están involucrados en un determinado ataque. Además, desde esta localización, es capaz de analizar el impacto en el sistema de cualquier acción, ya que tiene a su disposición toda la potencia del Sistema Operativo, y puede analizar detalladamente sistemas de ficheros, interfaces de red, etcétera, de forma ágil [E.04, Cro02, BM01, M.01b, D.01].

Sin embargo, en redes corporativas de medio o gran tamaño, el alto coste administrativo que suponen los procesos de instalación, configuración y mantenimiento de todo un parque de Sistemas de Detección de Intrusio-

nes orientados a la Máquina, hace que esta solución sea poco usada en la mayoría de las empresas. Por lo general su utilización quedaba circunscrita a sistemas servidores [Tea03, BM01], hasta la aparición de los sistemas SIEMs.

Ventajas

- **Posibilidad de determinar el éxito o fracaso de un ataque**

Habitualmente, los Sistemas de Detección de Intrusiones orientados a la máquina basan sus decisiones de análisis en los registros históricos de eventos del sistema o de servicios de aplicación, por lo que pueden obtener una medida precisa del grado de éxito o fracaso de un ataque. Los sistemas orientados a la red, por el contrario, solamente pueden detectar intentos de intrusión, pero no son capaces de determinar si dichos intentos han fructificado o no. De esta forma, en general, los sistemas orientados a la máquina presentan una tasa de falsos positivos muy reducida, lo que hace que sean un buen complemento de los sistemas de red: estos últimos detectan los primeros indicios de actividad sospechosa que aparecen en la red y sus homólogos orientados a la máquina comprueban si esa actividad ha tenido impacto en el sistema o no. [Tea03, T.02, BM01, Ho01, M.01b]

- **Monitorización de actividades específicas**

Este tipo de sistemas de detección puede monitorizar la utilización que los usuarios de un sistema hacen de sus recursos, pudiendo centrar su atención en cuestiones particulares como el acceso o modificación de archivos, utilización de credenciales y privilegios, manipulación de software, ejecución y manipulación de procesos, etcétera. Además, su especial ubicación hace posible que cualquier actividad sospechosa de poner en compromiso la seguridad del sistema pueda ser no sólo detectada, sino incluso interceptada, mediante una simple redirección de los servicios del Sistema Operativo en el que se encuentran instalados. [Tea03, Cro02, T.02, BM01, Ho01, M.01b, D.01, DDW99]

- **Detección de ataques físicos**

Evidentemente, un sistema de detección orientado a la máquina puede revelar cualquier actividad subversiva que se lleve a cabo mediante la presencia física del intruso ante la víctima. En aquellos casos en los que el atacante ha alcanzado tan importante nivel de acceso, un sistema orientado a la red es incapaz de responder, ya que es posible que el intruso no

necesite utilizar los recursos de comunicaciones de dicho sistema, o que incluso sea consciente de que no debe hacerlo para pasar desapercibido. Por el contrario, un sistema de detección orientado a la máquina no tiene ninguna dificultad para detectar dicha intrusión y notificarla convenientemente. [Ho01, D.01, K.99]

- **Detección de ataques a través de medios de comunicación cifrados**

Dado que este tipo de sistemas se instala directamente en la máquina a proteger, es posible dar respuesta a un problema que tradicionalmente se ha caracterizado por la ceguera absoluta que produce en los sistemas de detección orientados a la red, como es la cuestión del tráfico de red cifrado. Para poder analizar correctamente cualquier tipo de información, un sistema de detección orientado a la red debe ser capaz de comprender su estructura, lo cual puede convertirse en una tarea imposible de realizar en la práctica en el caso de que el atacante decida aplicar métodos de cifrado sobre dicha información. De esta forma, un intruso puede apoyarse en dichos algoritmos para conseguir eludir la vigilancia del sistema de detección. Sin embargo, dado que la información que llega al sistema atacado (conteniendo órdenes, archivos o herramientas) debe ser descifrada antes de ser interpretada por el Sistema Operativo, un sistema de detección orientado a la máquina no tiene dificultad alguna para realizar su cometido. [E.04, T.02, BM01, Ho01, D.01, DDW99, K.99, C.02]

- **Reducido tiempo de respuesta**

Dado que este tipo de sistemas sólo es responsable de analizar la actividad que se da en el sistema al cual dedican su vigilancia, el volumen de información que deben analizar es relativamente pequeño. Por ello, es posible proporcionar una respuesta en condiciones de tiempo real, o muy cerca de ellas. De esta forma, en la mayoría de los casos, la actividad del intruso puede ser detenida antes de que pueda producir cualquier daño en el sistema. [M.01b, D.01]

Inconvenientes

- **Elevado coste administrativo y económico**

En el caso de utilizar Sistemas de Detección orientados a la Máquina en una red corporativa con múltiples sistemas conectados, es necesario instalar, configurar, administrar y mantener un sistema de detección en cada

uno de dichos sistemas, llegando a adolecer problemas de *infoxificación*, o problemáticas a la hora de evaluar toda la información recibida. Evidentemente, esta consideración implica un extraordinario coste de administración, que a menudo hace que este tipo de soluciones sea descartado. Por el contrario, un único sistema orientado a la red es capaz, por sí solo o en pequeñas colectividades de agentes, de llevar a cabo la vigilancia de toda una red corporativa. Por ello, los costes económicos derivados tanto de los mencionados esfuerzos administrativos, como de la inversión económica propiamente dicha, son por lo general muy altos, en comparación con los sistemas orientados a la red. [E.04, Tea03, Cro02, BM01, M.01b, D.01, K.99]

- **Vulnerabilidad de emplazamiento**

Dado que este tipo de sistemas de detección se ubica en el propio sistema a proteger como si fuera un aplicativo convencional, en aquellos casos en los que el intruso consigue alcanzar cierto nivel de operatividad en dicho sistema, aparece el riesgo potencial de que el sistema de detección pueda ser desactivado o subvertido por dicho atacante. Por el contrario, un sistema de detección orientado a la red es susceptible configurarse de forma que su presencia sea no sólo inexpugnable sino incluso inadvertida para cualquier usuario. [Tea03, T.02, BM01, M.01b, D.01, DDW99, C.02]

- **Limitada área de influencia**

La propia concepción de estos sistemas hace que su ámbito de protección se vea limitado a un único sistema, debido a que solamente son capaces de detectar actividad subversiva a partir de los registros de solicitudes de servicio que los aplicativos hacen al Sistema Operativo de dicho sistema (ya sea en el nivel de aplicación o en el nivel de servicio), o a partir del tráfico de red exclusivamente dirigido hacia el mismo. Además, esta forma de actuación hace que el sistema de detección sea dependiente de la plataforma software (fundamentalmente del Sistema Operativo), lo que introduce una compleja componente de heterogeneidad en sistemas de información en los que conviven múltiples tipos de plataformas. Un sistema de detección orientado a la red, por el contrario, es capaz de detectar comportamientos ilegítimos dirigidos contra redes enteras, ya que la información que maneja fluye por el medio de comunicación y está rigurosamente estandarizada en forma de protocolos de comunicación. [Tea03, T.03a, Cro02, T.02, BM01, M.01b, K.99]

- **Sensibilidad del propio sistema de detección**

Un intruso potencial puede orquestar un ataque de denegación de servicio dirigido hacia el propio sistema de detección, con el objetivo de deshabilitarlo. De esta forma, una vez conseguida la desactivación de dicho sistema, las acciones de dicho intruso pueden realizarse con total impunidad e incluso sin dejar rastro. Ésta es una característica especialmente indeseada, tanto en este tipo de sistemas como en sistemas de detección orientados a la red, y que debe minimizarse haciendo especial énfasis en el desarrollo eficiente y optimizado de toda la arquitectura interna del sistema de detección. [T.02, BM01, M.01b, DDW99, K.99]

- **Alto consumo de recursos**

Dado que este tipo de sistemas de detección debe ubicarse en el propio sistema a proteger, necesita hacer uso de sus recursos para su ejecución. Por un lado, el consumo de recursos de almacenamiento suele verse incrementado notablemente, debido al volumen que habitualmente presentan los registros históricos de los sistemas operativos, así como al tamaño de los registros generados por el propio sistema de detección. Por otro, también es importante el consumo de capacidad de procesamiento que se requiere, ya que es necesario analizar cada una de las peticiones de servicio que los usuarios del sistema o sus aplicativos solicitan al Sistema Operativo local. Además, es necesario observar que, tal y como se ha explicado anteriormente, cada máquina de la red corporativa debe tener instalado su propio sistema de detección, por lo que esos grandes consumos se extienden a todo el parque informático de la organización. [E.04, T.02, BM01, DDW99, K.99, C.02]

2.1.3.1.2 Detección de Intrusiones orientada a la Red

La Detección de Intrusiones orientada a la Red constituyó una importante evolución, introducida por L. Todd Heberlein [HL90], del concepto de Detección de Intrusiones original. Hasta aquel momento la protección de sistemas informáticos se había realizado desde las propias infraestructuras hardware y software que se deseaban salvaguardar, y esta nueva filosofía originó un gran interés académico, y numerosas investigaciones científicas e inversiones comerciales. En general, este tipo de detección está considerado como el más poderoso y el que mejores prestaciones ofrece en grandes infraestructuras informáticas [M.01b]. Su ubicación, anexa al sistema

de explotación de la organización, a través de la captura continua del flujo de información transmitido a través de las redes corporativas, consigue que la interferencia o sobrecarga introducidas en el sistema sean nulas y que las decisiones generadas por el sistema de detección tengan carácter global [C.02]; todo ello sin los costosos requerimientos de administración de los sistemas orientados a la máquina. No obstante, estos sistemas también presentan algunas limitaciones, como por ejemplo su incapacidad para detectar ataques locales o para determinar el impacto de los ataques que detectan [BM01, D.01]. Por otro lado, el reducido impacto económico que tiene en los presupuestos de infraestructura de la organización, hace que la Detección de Intrusiones orientada a la Red sea una opción muy utilizada, y que alcanza en la actualidad altas cotas de cuota de mercado [Tea03]. En rasgos generales, éstas son las principales propiedades de este tipo de soluciones, aunque es importante observar sus características en conjunto. A continuación se relacionan las ventajas e inconvenientes que supone la utilización de esta clase de métodos.

Ventajas

- **Extensa área de influencia**

Un único Sistema de Detección de Intrusiones orientado a la Red es capaz de detectar por sí mismo cualquier tipo de actividad subversiva o anómala que utilice la infraestructura de comunicaciones de una organización como medio de transporte, incluso en redes con alto número de sistemas conectados. Además, en casos en los que el volumen de información que fluye a través del sistema de información supera la capacidad de respuesta del sistema de detección, su arquitectura general permite que el escalado de dicho sistema sea realmente sencillo y natural [D.01]. De esta manera, en el caso de grandes redes corporativas, la vigilancia de los recursos computacionales del sistema queda en manos no ya de un único sistema de detección, sino en una pequeña colectividad de procesos agente que colaboran en las tareas de recogida de información, análisis y respuesta. [Tea03, Cro02, T.02, BM01, M.01b, D.01, K.99]

- **Nulas interferencia y sobrecarga en el sistema objetivo**

El despliegue de este tipo de sistemas de detección tiene un impacto nulo sobre la infraestructura computacional y de comunicaciones subyacente,

ya que por lo general se destina para ello una máquina adicional dedicada en exclusiva. Así, habitualmente, un sistema orientado a la red captura de forma pasiva el tráfico de dicha red, sin intervenir activamente en ningún momento. Simplemente escucha, decide e informa si lo considera necesario. Por ello, esta solución es ampliamente adoptada por empresas y organizaciones de todo tipo [Tea03]; en perjuicio de los sistemas orientados a la máquina, que sí necesitan una importante cantidad de recursos provenientes del sistema a proteger, como pueden ser memoria principal, capacidad de almacenamiento, capacidad de procesamiento, etcétera. [E.04, T.02, BM01, DDW99, K.99, C.02]

- **Fortaleza de emplazamiento**

Un sistema de Detección de Intrusiones orientado a la Red puede configurarse de manera que sea virtualmente invisible para cualquier usuario de la organización, incluyendo por supuesto a cualquier potencial intruso que tenga como objetivo a dicha organización. Su carácter pasivo propicia esta característica e incluso es posible utilizar dispositivos hardware o componentes software que garantizan dicha propiedad [D03]. Existen ciertas soluciones de detección orientada a la red que abogan por un comportamiento reactivo ante las amenazas, pero mantienen igualmente esta propiedad de su inexpugnabilidad. Por el contrario, un sistema orientado a la máquina es fácilmente detectable por un intruso, quien puede intentar ponerlo en compromiso para conseguir sus fines. Además, esta invisibilidad evita cualquier posibilidad de alteración o corrupción de los recursos del sistema de detección mismo, de forma que los habituales métodos de eliminación de evidencias son inviables de antemano; a pesar de que existan ciertas técnicas [yNT98] dirigidas a evadir la vigilancia de dicho sistema de detección. [Tea03, T.02, BM01, M.01b, D.01, DDW99, C.02]

- **Detección de ataques orientados a la red**

Dado que este tipo de sistemas de detección analizan todos y cada uno de los paquetes de información que circulan por la red de comunicaciones, pueden detectar ataques que se realizan por debajo de los niveles de aplicación o de servicio a los que habitualmente acceden los usuarios de una máquina. Por lo general, un sistema orientado a la máquina no analiza todos los diferentes aspectos del tráfico de red, por lo que es incapaz de detectar numerosos tipos de ataque, como pueden ser los de denegación de servicio o los de fragmentación [yNT98]. Por el contrario, un sistema orientado a la red sí observa cada uno de los diferentes parámetros que componen dichos

paquetes, y puede responder de forma eficiente ante un mayor número de amenazas. [E.04, T.02, M.01b, DDW99, C.02]

- **Bajo coste administrativo y económico**

Debido a la notable amplitud del ámbito de influencia que por lo general poseen los sistemas orientados a la red, unas mínimas tareas administrativas y de supervisión y mantenimiento son en general suficientes para una explotación adecuada de dicho sistema de detección. Asimismo, la inversión económica necesaria se mantiene contenida en la mayoría de los casos, y es susceptible de incrementarse progresivamente a lo largo del tiempo en aquellas situaciones cuya evolución va requiriendo mayores prestaciones. [E.04, Cro02, BM01, M.01b, D.01, K.99]

- **Reducido tiempo de respuesta**

Al igual que los sistemas orientados a la máquina, los sistemas orientados a la red proporcionan por lo general excelentes tiempos de respuesta, lo que permite detectar un ataque mientras se está llevando a cabo. Esta capacidad de respuesta posibilita que el administrador reciba la notificación de dicho ataque a tiempo para responder convenientemente. Además, en el caso de que el sistema de detección incorpore capacidades de reacción, la respuesta puede en muchos casos llegar incluso a detener el ataque en cuestión. [BM01, Ho01]

- **Independencia de la plataforma**

Como se ha explicado anteriormente, este tipo de sistemas de detección utiliza los paquetes de información que se transmiten por la red para tomar sus decisiones; paquetes que se caracterizan por estar conformados en virtud de estándares que regulan sus diferentes características. Dado que dichos estándares tienen como objetivo la definición formal de protocolos de comunicación entre plataformas habitualmente heterogéneas, la información de la que se nutren los sistemas de detección orientados a la red es compatible e interpretable por un amplio número de plataformas hardware y software. Esto hace que un mismo sistema de detección sea capaz de advertir comportamientos subversivos en redes que interconectan sistemas de computación heterogéneos, lo que lo convierte en una herramienta de seguridad excepcional. Por el contrario, los sistemas orientados a la máquina son específicos de la plataforma, por lo que ante la necesidad de vigilancia de sistemas de información heterogéneos son necesarias diferentes soluciones de detección; con los costes administrativos y económicos que ello conlleva. [E.04, Cro02, T.02, M.01b, DDW99, C.02]

Inconvenientes

- **Rendimiento en redes de alta velocidad**

Ante el problema de la detección de intrusiones en redes de alta velocidad, los sistemas de detección orientados a la red adolecen por lo general de dificultades en el correcto procesamiento de todos los paquetes de información que fluyen por dichas redes, en situaciones de alta ocupación. Habitualmente, dichos sistemas integran en su arquitectura interna mecanismos de control de congestión para tratar de paliar este problema, pero, como es lógico, dichos mecanismos tienen capacidades finitas. Por ello, es posible la pérdida de paquetes de red que el sistema no ha sido capaz de capturar temporalmente para su análisis. Esta característica constituye una importante fuente de inspiración para todo tipo de intrusos, que intentan saturar el sistema de detección, incapacitándole para responder correctamente. Actualmente, la tendencia consiste en migrar el software de detección a dispositivos hardware que incrementen el rendimiento, así como la especialización de los tipos de ataque a detectar, y la optimización de los recursos computacionales necesarios para proporcionar una respuesta en unos márgenes de tiempo aceptables. A este respecto, está comúnmente aceptada la consideración de estos sistemas de detección orientados a la red como sistemas de tiempo real acríticos. [E.04, T.02, BM01, Ho01, M.01b, D.01, DDW99, K.99, C.02]

- **Problemas de detección de ataques a través de medios de comunicación cifrados**

Dado que los Sistemas de Detección de Intrusiones orientados a la Red necesitan acceder libremente a la información que circula por dicha red, cualquier obstáculo a ese acceso, ya sea físico como en el caso anterior, o lógico como en el caso en el que las comunicaciones estén sujetas a esquemas de cifrado de la información, incapacita a dichos sistemas para cumplir con su cometido. Un sistema de detección que no es capaz de interpretar la información contenida en un paquete de red, no puede realizar ningún análisis respecto de dicho paquete, ni tomar decisión alguna. Éste es uno de los grandes handicaps que tradicionalmente ha venido sufriendo la Detección de Intrusiones orientada a la Red, si bien existe una evolución de estos sistemas por la cual es posible solventar este problema, y que se conoce como Detección de Intrusiones orientada al Nodo de Red[1]

[1] Este enfoque da lugar a los conocidos Sistemas de Detección de Intrusiones de Nodo de Red o Network Node Intrusion Detection Systems.

[D03, Cro02, M.01a]. Este enfoque consiste en convertir un sistema orientado a la red en un sistema orientado a la máquina, que analice el tráfico proveniente del subsistema de comunicaciones de dicha máquina, después de haber sido descifrado por la capa criptográfica de la pila de protocolos de red. Por supuesto, este avance implica un retroceso en la tradicional capacidad de los sistemas orientados a la red de detectar intrusiones en grandes redes. [E.04, T.02, C.02, BM01, Ho01, D.01, DDW99, K.99]

- **Imposibilidad de determinar el éxito o fracaso de un ataque**

Por lo general, los sistemas de detección orientados a la red no son capaces de determinar si un ataque detectado por ellos ha sido exitoso para el atacante o no. Sólo pueden precisar que en un momento dado se produjo dicho ataque. Esta particularidad obliga a que, después de detectado un determinado ataque, el administrador del sistema deba investigar manualmente el impacto que ha podido tener dicho ataque en su organización. [Tea03, T.02, BM01, Ho01, M.01b]

2.1.3.2 **Análisis**

La fase de análisis constituye el núcleo central del modelo general de funcionamiento de todo sistema de detección, ya que es en él donde deben tomarse las decisiones importantes que afectan a la seguridad del sistema de información que se pretende proteger. La anterior fase de recogida de información, así como la siguiente, de respuesta, están caracterizadas por realizarse de forma prácticamente automática, mecánica. Por el contrario, en esta fase de análisis es necesario disponer de un modelo de representación de conocimiento y de un mecanismo de inferencia de conclusiones que sean capaces de procesar las complejas relaciones que existen entre los múltiples parámetros que es necesario estudiar. Por supuesto, es en esta fase de análisis donde se encuentra concentrado el mayor interés estratégico de todo el área de conocimiento de la Detección de Intrusiones, y es en ella también donde se registra una mayor actividad investigadora.

A continuación se describen las principales características de las dos grandes filosofías de funcionamiento sobre las cuales está construida la mayoría de los componentes de análisis de los actuales Sistemas de Detección de Intrusiones: Detección de Usos Indebidos y Detección de Anomalías. Como se comprobará, ambos métodos proporcionan importantes beneficios, pero no consiguen evitar ciertas debilidades. Esta situación obliga a seguir investigando en búsqueda de un modelo que sea capaz de aunar las

ventajas de ambos métodos, minimizando en la medida de lo posible los inconvenientes [CAT05, E.04, Tea03, Gol03, HT03, T.02, BM01, Ho01, D.01, ACFW00, Axe00, T.00].

2.1.3.2.1 Detección de Usos Indebidos

Actualmente, la Detección de Usos Indebidos es el enfoque más extendido y el que ha fructificado con mayor firmeza en el mercado [Tea03, C.02, BM01]; éxito que ha venido producido fundamentalmente por su eficiencia y sencillez de administración. Su principio de funcionamiento es simple: a partir del conocimiento de los diferentes tipos de ataques utilizados por la comunidad hacker, basta con recoger los eventos que se produzcan en un determinado sistema y contrastarlos contra dicho conocimiento, para poder detectar intentos de subversión en el sistema.

Por lo general, este conocimiento se estructura en forma patrones que describen las peculiaridades de cada ataque. De esta forma, una base de conocimiento puede estar compuesta por varios miles de patrones o firmas [BP04, AAMP03], cada uno de los cuales debe ser comparado con los sucesos que van teniendo lugar en la explotación cotidiana del sistema de producción. Este modo de operación hace que el sistema sea realmente sólido y que detecte con precisión toda actividad maliciosa que tenga registrada en su base de firmas. Sin embargo, es precisamente este mismo modo de operación la causa de su más importante limitación: no es posible detectar nada que no esté previamente documentado.

De esta manera, un intruso (que puede estar al corriente de la configuración de las bases de conocimiento que se estén utilizando) puede realizar una ligera modificación de un ataque y pasar desapercibido para el sistema de detección. Es más, cada día aparecen ataques completamente nuevos, que por supuesto pasan inadvertidos a los detectores, a menos que se incluyan las nuevas firmas. Como es obvio, esta cuestión obliga a realizar un continuo y costoso proceso de mantenimiento, que queda en manos del operador humano, y que hace que el sistema de detección vaya perdiendo progresivamente su eficiencia. Ante continuas actualizaciones de la base de firmas, llega un momento en el que el volumen de patrones a contrastar consume más tiempo del que tardan en producirse los eventos del sistema. El detector se satura y se empiezan a ignorar dichos eventos. Por todo ello, si se desea conseguir una auténtica protección ante ataques novedosos, es necesario explorar otras vías de investigación que complementen a los sistemas basados en firmas. [E.04, Gol03, D03, HT03, BM01, Axe00, DDW99, K.99]

A continuación se relacionan las ventajas e inconvenientes fundamentales que presenta este tipo de sistemas de detección:

Ventajas

- **Precisión en la detección**

Los Sistemas de Detección de Usos Indebidos son muy efectivos en la detección de los ataques que tienen documentados. Esto es debido a que su principio de funcionamiento se reduce a la comparación de la actividad cotidiana del sistema que se pretende proteger, con una base de firmas de ataques que se construye a partir de vasto conocimiento experto sobre métodos de intrusión y procedimientos para explotar vulnerabilidades conocidas. De esta forma, cualquier actividad subversiva que se encuentre tipificada en dicha base de conocimiento es advertida rápidamente. [E.04, Gol03, HT03, BM01, Axe00, DDW99]

- **Ausencia de falsos positivos**

Dada la simplicidad del método de detección, no es habitual que se produzca el fenómeno del falso positivo, siempre y cuando la base de conocimiento se encuentre convenientemente configurada. Ésta es precisamente la mayor ventaja que presenta este tipo de sistemas de detección con respecto a otros, como puede ser la Detección de Anomalías, que ha sufrido tradicionalmente este problema. [E.04, D03, BM01, ACFW00, DDW99, LP99]

- **Estabilidad del conocimiento**

Una característica negativa que presentan algunos Sistemas de Detección de Intrusiones más ambiciosos que los basados en la Detección de Usos Indebidos, es la posibilidad de que los modelos de representación de conocimiento adaptativos que utilizan puedan ser dirigidos progresiva e intencionadamente[1] para que, llegado el momento, dejen pasar desapercibidos ciertos eventos que el potencial intruso no desea que sean detectados. Si el sistema de detección es capaz de adaptarse a los cambios de comportamiento de los usuarios, es posible que alguno de esos usuarios utilice esa indeseable característica con fines maliciosos. Sin embargo, en el caso de la detección de firmas, la sencilla estructura de su motor de decisión elimina este problema. [E.04, D03, BM01, DDW99, K.99]

[1] Dicha técnica es conocida habitualmente como *session creeping*.

- **Bajo coste administrativo**

Un Sistema de Detección de Usos Indebidos puede proporcionar de forma rápida y fiable un diagnóstico exacto sobre la situación de un sistema de información, sin necesidad de laboriosos procedimientos administrativos y sin necesidad de disponer de personal de seguridad excepcionalmente experimentado. [E.04, Gol03, BM01, ACFW00, DDW99, LP99]

Inconvenientes

- **Incapacidad de detección de ataques no documentados**

La limitación por excelencia de todo sistema de detección de usos indebidos, dada la naturaleza del propio principio de funcionamiento, consiste en la incapacidad absoluta de este tipo de sistemas de advertir cualquier comportamiento subversivo que no se encuentre documentado en su base de conocimiento. Dado el extraordinario número de vulnerabilidades que se encuentran diariamente, así como de procedimientos de ataque que las explotan, esta cuestión adquiere de forma natural carácter de urgencia. Actualmente, las soluciones al problema pasan por la actualización automática periódica de las bases de firmas, si bien este método termina por reducir la eficiencia del sistema de detección a medio y largo plazo. [E.04, D03, K.03, BM01, ACFW00, Axe00, DDW99, K.99, LP99]

- **Rendimiento decreciente**

El crecimiento que se produce en las bases de conocimiento a medida que se detectan, analizan y documentan nuevos tipos de ataques, hace que dichas bases sean complejas de mantener, y que el rendimiento del sistema termine por degradarse. En esta situación, el sistema de detección puede llegar incluso a saturarse, y a tomar la decisión de ignorar aquellos eventos que están saturando sus mecanismos de control de la congestión. Por supuesto, si el sistema se ve obligado a dejar de analizar eventos, su eficacia pasa a verse en un serio compromiso. De hecho, existen técnicas de evasión de Sistemas de Detección de Intrusiones [yNT98] que se basan en esta indeseada característica. [E.04, Gol03, DDW99, K.99]

2.1.3.2.2 Detección de Anomalías

Con el objetivo de aportar una solución al principal problema de los Sistemas de Detección de Usos Indebidos, y proporcionar un método capaz

de detectar ataques desconocidos hasta el momento, surge la Detección de Anomalías [Den87], si bien hasta la fecha existen pocos casos de éxito de dicha tecnología, debido fundamentalmente a la complejidad de administración que la ha venido caracterizando. Su principio de funcionamiento consiste en primer lugar en elaborar un perfil del comportamiento habitual de los usuarios. A continuación, la actividad cotidiana de los mismos es comparada con dicho perfil, y en el caso de que se observe alguna desviación significativa, dicha actividad será clasificada como anómala. Por último, se generará la pertinente respuesta, que en función de la configuración del sistema podrá quedar reducida a una notificación de alarma al administrador, o por el contrario llegar a provocar un efecto reactivo sobre el sistema.

Por lo general, este modelo de funcionamiento está basado en las propiedades estadísticas de los diferentes eventos que se producen en el sistema. De esta forma, es posible que el sistema de detección aprenda la naturaleza del comportamiento de los usuarios del sistema, e infiera sus conclusiones, por lo general construidas en base al concepto de probabilidad. Este modo de operación hace que el sistema pueda prescindir de todo conocimiento apriorístico, y que pueda detectar ataques para los cuales no existe un modelo previamente documentado. Sin embargo, su naturaleza probabilística y la necesidad del sistema de detección de permanecer en un estado de adaptación continua a las variaciones del comportamiento de los usuarios, hace a dichos sistemas susceptibles de generar falsos positivos. Esta característica no es preocupante siempre que se mantenga dentro de unos límites de excepcionalidad, pero puede convertirse en un serio problema si se da con mayor frecuencia. De hecho, la minimización de los falsos positivos es uno de los mayores retos que actualmente presenta esta tecnología [E.04, D03, BM01, ACFW00, Axe00, DDW99, K.99, LP99].

A continuación se relacionan las principales ventajas e inconvenientes que presenta la Detección de Anomalías:

Ventajas

- **Detección de ataques desconocidos**

La mayor capacidad de este tipo de Sistemas de Detección de Intrusiones es precisamente la de poder responder ante lo desconocido. Dado que su filosofía de funcionamiento está basada en el aprendizaje que el propio sistema

de detección hace sobre la realidad de la infraestructura hardware y software que pretende proteger, es este conocimiento aprendido el elemento que marca la diferencia entre lo legítimo y lo subversivo. De esta forma, no es necesario documentar todos y cada uno de los tipos de ataque que aparecen diariamente, sino que basta con realizar un ejercicio de introspección que determine cómo se comporta habitualmente un sistema de información, y cómo no. [E.04, Gol03, D03, BM01, ACFW00, Axe00, DDW99, K.99, LP99]

- **Documentación de ataques desconocidos**

Dada la poderosa propiedad anterior, un sistema de detección basado en anomalías puede generar automáticamente información descriptiva sobre los nuevos ataques que detecta. De esta forma, es posible construir, en base a dicha información, nuevas firmas de ataques que pueden ser utilizadas por los sistemas de detección de usos indebidos, de cara a la utilización conjunta de ambos. [BM01]

- **Rendimiento constante**

El modelo de representación de conocimiento de este tipo de sistemas no necesita incrementar su volumen de conocimiento, sino simplemente adaptarlo a ajustarlo a la realidad de la infraestructura subyacente. Esta propiedad hace que estos sistemas tengan un prometedor futuro, ya que no adolecen del problema de la pérdida de eficiencia que sufren los modelos de detección de usos indebidos. [E.04, Gol03]

Inconvenientes

- **Inestabilidad del conocimiento**

Dada la necesidad de este tipo de sistemas de detección de adaptación continua a la siempre cambiante realidad del sistema a proteger, el conocimiento adquirido por los Sistemas de Detección de Anomalías presenta cierta propensión a la inestabilidad. El comportamiento del sistema cambia, y ese cambio puede influir para que lo que en un momento dado ha podido ser considerado como legítimo, ahora pase a ser subversivo, y viceversa: que lo que en el pasado era considerado anómalo, pase a convertirse en habitual. Por supuesto, esta cuestión se da cerca de los umbrales de detección, y no significa que el sistema sufra de indeterminismo. [E.04, BM01, Axe00, DDW99, K.99, LP99].

- **Posibilidad de subversión del conocimiento**

Debido a la misma necesidad de adaptación del apartado anterior, aparece una remota, aunque posible, probabilidad de que el inherente proceso de aprendizaje de este tipo de sistemas sea utilizado subversivamente, con el objetivo de que en un momento del futuro el sistema de detección deje de percibir como anómalos ciertos eventos maliciosos. Para ello, un intruso puede provocar a lo largo de un cierto periodo de tiempo ciertos sucesos normales (legítimos), pero cuyos atributos cada vez se van acercando progresivamente al terreno de lo anómalo. De esta forma, teóricamente, llegaría un momento en el futuro en el que el sistema, que se habría adaptado a esos cambios, consideraría un evento antes subversivo como perfectamente legítimo. [E.04, D03, BM01, DDW99, K.99]

- **Alto coste administrativo**

En la actualidad, el proceso de aprendizaje que caracteriza a los sistemas de detección de anomalías aún no ha conseguido liberarse de una relativamente costosa tarea de configuración y administración. Asimismo, y dada la naturaleza cambiante de dichos sistemas, el mantenimiento de los mismos también contribuye de forma notable a incrementar los esfuerzos que una organización necesita soportar para que su sistema de detección pueda ser explotado con las mayores garantías. Además, dicho proceso de aprendizaje no sólo precisa de esas importantes tareas de configuración y administración, sino que, además, los modelos computacionales que se utilizan (como pueden ser las redes neuronales o los algoritmos genéticos) requieren muy frecuentemente la supervisión del mismo por parte del operador humano. [E.04, BM01, ACFW00, K.99]

- **Presencia de falsos positivos**

Debido fundamentalmente a la naturaleza probabilística de los modelos de Detección de Anomalías, es frecuente la necesidad de tomar un compromiso sobre la magnitud del volumen de respuestas que se consideran convenientes para su correcto procesamiento posterior. Dicho compromiso se establece habitualmente en base a valores umbral que separan lo que se considera normal (y por lo tanto, legítimo) de lo que se considera anómalo (y por lo tanto, ilegítimo). Un umbral muy restrictivo hace que cualquier desviación, por leve que sea, sea considerada merecedora del lanzamiento de una alarma u otra acción de respuesta. En principio,

la seguridad del sistema es muy alta, y se responde ante cualquier evento sospechoso. Sin embargo, el volumen de respuestas es enorme, y en el caso habitual de utilizarse la simple respuesta pasiva mediante notificaciones de alarma, el administrador queda automáticamente saturado de ingentes cantidades de mensajes que no puede procesar y que por lo general sólo implican acciones legítimas pero ligeramente desviadas del perfil de comportamiento. O lo que es lo mismo: falsos positivos. Además, y lo que es aún peor, entre esos grandes volúmenes de mensajes suelen pasar desapercibidos auténticos peligros, los cuales no pueden ser físicamente diferenciados de los anteriores. Por el contrario, un umbral muy permisivo genera un número razonable de respuestas, pero posibilita la no detección de ataques cercanos a dicho umbral, dando origen a los aún más peligrosos falsos negativos. Por todo ello, parece evidente la necesidad de buscar un equilibrio entre los dos extremos, que permita minimizar ambos efectos indeseados, si bien su eliminación completa es improbable. [E.04, Gol03, D03, BM01, ACFW00, Axe00, DDW99, K.99, LP99]

2.1.3.3 **Respuesta**

Una vez que el sistema de detección ha recogido un determinado evento y tomado la decisión que considera más apropiada sobre él, es necesario dar una respuesta oportuna al mismo. En la mayoría de los casos, dicha respuesta implica el lanzamiento de un mensaje de alarma hacia una consola de gestión a cargo del administrador de dicho sistema. Este modo de actuación es lo que se conoce como Respuesta Pasiva, y se realiza habitualmente de forma remota, bien mediante un protocolo de notificación propio del sistema de detección, bien mediante algún esquema estándar de gestión de sistemas como puede ser por ejemplo SNMP[1] [Tea03, Gol03], o bien mediante los servicios de notificación de eventos que proveen los sistemas operativos. Por el contrario, existen otras soluciones que apuestan por el uso de técnicas más activas, y que intentan mitigar de forma automática los efectos de los potenciales ataques. Por lo general, una vez detectado un ataque, este tipo de aproximaciones intentan recabar toda la información posible sobre el origen de dicho ataque, tanto para poder así orquestar una respuesta más precisa, como para poder tomar las acciones legales correspondientes con mayor conocimiento de causa. Además, también es habitual llevar a cabo acciones correctoras del entorno atacado, abortando las cone-

[1] SNMP o Simple Network Management Protocol: Protocolo de gestión de red, basado en UDP [R.00].

xiones del intruso con la víctima, bloqueando el acceso de dicho intruso, protegiendo el acceso a servicios del sistema agredido, etcétera. Asimismo, existen sistemas de detección que presentan un mayor nivel de agresividad, y optan directamente por contraatacar contra el origen de la amenaza. Por último, como se ha explicado en apartados anteriores, es importante prestar atención a un nuevo enfoque que está surgiendo al respecto, y que coloca al sistema de detección dentro del flujo de información, de forma que le es posible no ya sólo detectar un cierto ataque, sino incluso interceptarlo. Este enfoque se denomina Prevención de Intrusiones, y pretende proporcionar un mayor nivel de seguridad, filtrando de forma selectiva aquellos eventos dirigidos al sistema que se pretende proteger. [E.04, Tea03, D03, HT03, T.03a, T.02, C.02, BM01, ACFW00, Axe00, DDW99, K.99] A continuación se describen las dos principales filosofías de respuesta que presentan los actuales sistemas de detección: La respuesta pasiva y la respuesta activa.

2.1.3.3.1 Respuesta pasiva

Como se ha descrito anteriormente, la respuesta pasiva constituye la forma de respuesta más simple y a su vez más extendida, fundamentalmente debido a dos razones. (1) Por un lado, es habitual que los administradores de sistemas de detección de intrusiones prefieran mantener un control estricto sobre las acciones que se llevan a cabo en el sistema de información que tienen a su cargo, de forma que cualquier hipotética decisión que pudiera tomar dicho sistema de detección representa un riesgo de inestabilidad en el sistema que en la mayoría de los casos no es aconsejable tomar automáticamente, sin un cierto proceso de reflexión. (2) Por otro lado, parece lógico que la responsabilidad de llevar a cabo acciones críticas para el sistema, como suelen ser las relativas a modificaciones en la política de seguridad (como es lo habitual), recaigan en un operador humano, y no en un sistema que se encuentra en continuo peligro de ser subvertido. [E.04, BM01].

A continuación se relacionan las ventajas e inconvenientes más importantes de este método de respuesta:

Ventajas

- **Rapidez de respuesta**

La principal característica positiva de este método de respuesta es su sencillez y rapidez. Es posible lanzar una notificación de alarma a la consola

de gestión del administrador o incluso a su teléfono móvil en el mismo instante que se detecta un ataque. Una correcta configuración del sistema de detección, que mantenga la tasa de falsos positivos en unos límites razonables, en combinación con este tipo de respuesta, puede ser perfectamente válido en la mayoría de los casos. [ACFW00, K.99]

- **Imposibilidad de subversión**

Al contrario que en otros tipo de sistemas más ambiciosos y agresivos, con este tipo de respuesta no es posible que un potencial intruso llegue a utilizar subversivamente las peligrosas capacidades de actuación sobre el entorno de dichos sistemas. Deberá ser el operador humano quien decida si es necesario activar el registro de datos sobre el intruso, bloquear su acceso al sistema, o contraatacarle. Es un método simple, pero no adolece de los delicados efectos laterales de otros esquemas más sofisticados. [Tea03, D03, T.03a, C.02, BM01, ACFW00, Axe00, K.99]

Inconvenientes

- **Incapacidad de reacción**

Debido a la propia naturaleza de este método de respuesta, no es posible realizar ningún tipo de operación ante una determinada evidencia de agresión. En algunos casos, existen ciertas acciones operativas que pueden tomarse sin riesgo de ser utilizadas en beneficio del atacante, como por ejemplo el filtrado de paquetes de red mal formados, y que este método no puede realizar por defecto. [T.03a, BM01, K.99]

- **Posibilidad de saturación del operador humano**

En el caso de disponer de un sistema de detección con respuesta pasiva que sufra del problema de los falsos positivos, el método por defecto consistente en el envío de notificaciones de alarma al administrador puede llegar a saturarle por completo. De hecho, puede llevarle hasta la más absoluta inoperancia. Además, en muchos casos un determinado evento malicioso es el desencadenante de otros muchos, de forma que un único ataque se traduce en una avalancha de notificaciones. [E.04, T.03a, K.99]

2.1.3.3.2 Respuesta activa

Como se ha explicado anteriormente, existen tres niveles de agresividad en las respuestas de tipo activo: recogida adicional de información, reconfiguración del entorno lógico de la víctima, y contraataque [E.04, Tea03, D03, T.03a, BM01, ACFW00, Axe00, DDW99, K.99]. Además, existe un aspecto adicional, en función de la ubicación del sistema de detección, que permite no ya sólo detectar un determinado ataque, sino además prevenir al sistema contra él [T.04, E.04, HT03, T.03b, LP99]. En el primer caso, el principio de funcionamiento consiste en, dado que habitualmente el registro de información que realiza el sistema de detección es muy limitado por cuestiones de espacio, incrementar la granularidad de la información que se recaba, una vez que se tiene constancia de estar sufriendo un ataque. De esta forma, es posible registrar las direcciones origen del ataque, los identificativos de usuario responsables del mismo, los recursos que están siendo utilizados, etcétera, de cara a una posterior depuración de responsabilidades. En segundo lugar, se plantea la cuestión de intentar detener un ataque, una vez detectado, o al menos, evitar ataques posteriores. Para ello, la solución habitual pasa por la terminación abrupta de las conexiones de red que el intruso ha activado hacia la víctima. Es posible que el mal ya esté hecho, pero así se intenta al menos detener cualquier actividad del agresor lo antes posible. Además, también es habitual intentar evitar cualquier intento futuro de acceso por parte de dicho intruso, mediante la reconfiguración de los dispositivos de enrutado y de filtrado de paquetes. Asimismo, una solución de agresividad máxima consiste en atacar mediante patrones de ataque predefinidos aquellas direcciones desde las cuales se lanzó el mencionado ataque. Por supuesto, este último extremo no es recomendable en la práctica, dadas las ambigüedades legales existentes. Por último, en los casos en que el sistema de detección se encuentre ubicado dentro del flujo de información por el que deben circular las órdenes de los usuarios hacia el sistema, es posible realizar filtrado de alta precisión, exclusivamente de aquellos eventos que se identifican como potencialmente peligrosos.

Ventajas

- **Reacción automática**

En este tipo de sistemas es posible configurar un cierto grado de respuesta automática, de forma que es posible graduar su intensidad, en función del grado de certidumbre de que se disponga. Por ejemplo, aquellos eventos de

naturaleza evidentemente maliciosa pueden ser filtrados directamente por el sistema de detección, mientras que otras cuestiones en las que exista una mayor ambigüedad pueden seguir dejándose en manos del administrador. Por su parte, otros eventos de riesgo medio pueden activar un mecanismo de registro detallado. [E.04, D03, T.03a, BM01, ACFW00, K.99]

- **Reducción del volumen de notificaciones de alarma**

A pesar de que esta cuestión depende en gran medida de la naturaleza de la fase de análisis, una respuesta activa configurada convenientemente, de forma gradual, puede contribuir a reducir notablemente la carga de trabajo del operador del sistema de detección. [D03, T.03a]

- **Posibilidad de intercepción del ataque**

En el caso de que el sistema de detección se encuentre ubicado en forma de sistema de prevención de intrusiones, esto es, dentro del flujo de información y con capacidad de intercepción de cada uno de los eventos que componen dicho flujo, a las capacidades anteriores es posible añadir esta última. Con ella, el objetivo tradicional de detección de la intrusión se extiende extraordinariamente, y hace posible que cualquier evento clasificado como malicioso sea eliminado del sistema que se está protegiendo. [BM01, ACFW00, K.99]

Inconvenientes

- **Potencial exceso de responsabilidad**

Un importante inconveniente de los métodos de respuesta activa es precisamente una de sus capacidades. Así como la absoluta carencia de responsabilidad del sistema de detección perjudica el rendimiento de dicho sistema, un alto grado de responsabilidad supone un riesgo de inestabilidad difícilmente aceptable por cualquier administrador experimentado. De esta forma, la solución lógica pasa por encontrar un equilibrio entre ambos extremos, de manera que el sistema pueda hacerse cargo de cuestiones menores y de repercusión controlada, mientras que el operador humano toma la responsabilidad de las cuestiones de mayor impacto. [Tea03, D03, T.03a, BM01, ACFW00, Axe00, K.99]

- **Posibilidad de subversión**

Esta característica se encuentra muy relacionada con la anterior, ya que en aquellos casos en los que el sistema de detección asume un gran nivel de responsabilidad, es posible que esa capacidad de actuación sea utilizada subversivamente por el intruso. Por ejemplo, es habitual la reconfiguración de dispositivos de red para que realicen el bloqueo de aquellas direcciones de red desde las cuales se ha detectado un determinado ataque. De esta forma, un agresor que falsifica dichas direcciones en los paquetes de información que intercambia con su víctima puede conseguir que el propio sistema de detección se encargue de denegar un determinado servicio a todas las máquinas de la red que está protegiendo, con lo que habrá conseguido servirse a su antojo de la herramienta que supuestamente debía velar por la seguridad del sistema de información. [Tea03, D03, T.03a, C.02, BM01, ACFW00, Axe00, K.99]

2.2 NIDS. Sistemas de Detección de Intrusiones basados en Red.

Las actuales tendencias investigadoras que se dan en el área de los Sistemas de Detección de Intrusiones basados en Red, se basan en el uso de modelos de representación e inferencia que apliquen alguna de las acepciones del concepto de certidumbre [CGH97, H.04a, S.12], como herramienta mediante la cual superar los grandes retos existentes que vienen de lejos y no han sido completamente superados [M.03b, BWC02, T.02, BM01, BCH+01, LSC+01, K.01b, D.01, ACFW00, Axe00, ySR00, T.00, MJ00, M.00, DDW99, K.99, LP99, yNP97, MB04, Den87, BM01, Bur05, CAT05, PS05, E.04, Tea03, D03, K.03].

En este capítulo se describe el entorno de trabajo sobre el cual se desarrolla la actividad investigadora del presente estudio, tanto en lo relativo a la cuestión de la Detección de Intrusiones. Durante la década de los ochenta y hasta la actualidad, el crecimiento exponencial de la actividad subversiva contra sistemas de información de todo tipo ha suscitado un desarrollo igualmente acelerado en la investigación que se ha venido desarrollando en el área de la Detección de Intrusiones.

Tradicionalmente, las tareas de auditoría de los eventos de seguridad que se producían en un determinado sistema de información eran llevadas a cabo de forma manual. Las máquinas en explotación existentes en los diferentes tejidos de la sociedad no eran numerosas, las cifras de usuarios y aplicaciones se mantenían en niveles bajos, y era no era descabellado el hecho de que dichas tareas se realizaran de esa manera. Sin embargo, a medida que el número de máquinas, usuarios, servicios de aplicación, capacidades de conectividad, etcétera, crecía, la magnitud del número de eventos de seguridad adquiría progresivamente una mayor dimensión que hacía inviables los tradicionales procedimientos de auditoría. Era necesario abordar la búsqueda de nuevos métodos de trabajo automatizados [D03, Cro02, BM01, P.01, ACFW00].

2.2.1 Técnicas de análisis

Como se ha expuesto anteriormente, en los apartados anteriores, el componente de Análisis constituye la línea de investigación en la que existe actualmente un mayor interés y sobre la cual se está llevando a cabo la mayor actividad investigadora, tanto dentro del mundo académico como dentro de la industria de la Seguridad de la Información.

Dicho componente acapara en estos momentos el mayor número de publicaciones de las más prestigiosas conferencias internacionales, y centra sobre sí mismo las mayores inquietudes innovadoras del área de conocimiento de la Detección de Intrusiones.

De esta manera, con el objetivo de situar la actividad investigadora realizada en el presente estudio en el vasto horizonte de cuestiones propias del área de la Detección de Intrusiones, se describen a continuación las principales filosofías de análisis existentes en la actualidad, esto es: Detección de Usos Indebidos y Detección de Anomalías.

2.2.1.1 **Detección de Usos Indebidos**

La Detección de Usos Indebidos hace referencia al método de Detección de Intrusiones que aboga por utilizar la definición previa de dichas intrusiones para su posterior contraste con la actividad cotidiana del sistema que se desea proteger [BM01]. De esta manera, la operatoria del sistema de detección está basada por completo en una masiva recopilación de conocimiento experto proveniente del operador humano, y que incluye una ingente cantidad de definiciones de ataques conocidos, así como de vulnerabilidades conocidas. Así pues, una vez recogido dicho conocimiento experto, la detección de cualquier rastro de una actividad que previamente haya sido catalogada como maliciosa (y que utilice por lo tanto alguno de los ataques documentados, o intente explotar alguna de las vulnerabilidades conocidas) puede llevarse a cabo de forma eficiente y muy precisa.

No obstante, es precisamente el motivo de su efectividad la razón de que este tipo de Sistemas de Detección de Intrusiones se encuentre enormemente limitado para responder ante situaciones de riesgo para las que no existe un modelo previamente documentado. Por lo general, ante una ligera variación de uno de los ataques documentados, el sistema de detección no localiza en su base de conocimiento la firma o patrón representativo de dicho ataque y es por lo tanto incapaz de responder. Esta cuestión obliga al administrador humano a una minuciosa y costosa elaboración de las firmas de ataque, con el doble objetivo de que sean lo más genéricas posible y de que no generen falsos positivos. Por otro lado, la respuesta ante un Zero-Day Attack es igualmente fallida, lo que obliga (al igual que en el caso anterior) a una periódica actualización de dicha base de conocimiento. En general, estas características propias de los sistemas de detección basados en Detección de Usos Indebidos hacen que dichos sistemas sean calificados habitualmente como precisos pero incompletos [DDW99].

Así pues, con el objetivo de proporcionar una visión general sobre el funcionamiento interno de este tipo de sistemas de detección, que permita extraer las conclusiones necesarias para identificar las problemáticas existentes y enunciar los grandes retos que aún presenta esta tecnología, se describen a continuación los principios básicos de funcionamiento sobre los cuales está construida la inmensa mayoría de ellos.

2.2.1.1.1 Reconocimiento estático de Patrones.

Está técnica es la forma de Detección de Usos Indebidos más simple que existe, así como la más extendida, y está basada en la búsqueda de patrones o firmas básicas[1] dentro del flujo de eventos proporcionado por el mecanismo de recogida de información [AAMP03, SM88]. Dichos patrones están constituidos por lo general por un conjunto de atributos discretos que tiene el objetivo de formalizar el conocimiento experto que se deberá utilizar durante el proceso de detección. Por otro lado, a pesar de las limitaciones semánticas que presenta este método, su sencillez permite conseguir una gran eficiencia a la hora de tomar decisiones, por lo que su utilización es en muchos casos más que suficiente. Algunos de los actuales Sistemas de Detección de Intrusiones que utilizan, entre otras, esta técnica son Snort [R+99], SURICATA [sur10] , Bro96[Pax98], NFR97 [RM97], RealSecure, CISCO Secure IDS [CIS06], o Dragon [Inc06a].

2.2.1.1.2 Probabilidad Condicional

El método anterior presenta un comportamiento suficiente en muchos casos, pero adolece de una importante carencia de capacidad representativa [AAMP03]. En concreto, es habitual el hecho de que un determinado patrón identifique la existencia de su ataque correspondiente no de forma categórica, sino con una cierta probabilidad [S.95]. De esta manera, es posible la aparición de situaciones en las que la detección del patrón no implique absolutamente el hecho de que el sistema esté siendo atacado. Por ello, un motor de análisis semánticamente completo debe contemplar el concepto de probabilidad condicional [H.04a], ecuación 2.1[2]

[1] Es muy habitual la definición de patrones mediante la especificación de cadenas de símbolos que se corresponden con diferentes métodos de ataque. Si se localiza una cadena o subcadena concreta dentro de un determinado evento, se habrá detectado el correspondiente ataque.

[2] La expresión indica la probabilidad condicional de que se haya producido una intru-

$$P(Intrusion/Patron) \tag{2.1}$$

A continuación, mediante la aplicación del Teorema de Bayes [Uni15] a la expresión anterior, es posible obtener, la ecuación 2.2:

$$P(Intrusion/Patron) = P(Patron/Intrusion)\frac{P(Intrusion)}{P(Patron)} \tag{2.2}$$

De esta forma, a partir de la ecuación 2.2, un administrador humano es capaz de cuantificar dicha probabilidad condicional, simplemente mediante el cálculo de los tres términos que componen dicha expresión. Para ello, es necesario disponer de un registro histórico que recoja la experiencia del sistema, y a través del cual sea posible cuantificar la probabilidad a priori [H.04a] de la ocurrencia de una intrusión, o $P(Intrusion)$.

Asimismo, gracias a dicho histórico también es posible determinar, para cada uno de los patrones de ataque, tanto la probabilidad a priori de dicho patrón, o $P(Patron)$, como la probabilidad condicional $P(Patron/Intrusion)$.

Por otro lado, este modo de calcular la probabilidad condicional puede no ya utilizarse ante eventos individuales, sino extenderse incluso a secuencias de eventos, con lo que se consigue una mayor capacidad representativa del estado del sistema que se pretende proteger. Para ello, sería necesario calcular, de una forma similar, la probabilidad [S.95] (ecuación 2.3).

$$P(Intrusion/Secuencia\ De\ Eventos) \tag{2.3}$$

2.2.1.1.3 Sistemas Expertos

Otra filosofía de análisis, extremadamente potente, que se ha aplicado en algunas ocasiones a la Detección de Intrusiones basada en Detección de Usos Indebidos es la utilización los denominados Sistemas de Producción o Sistemas Expertos[yRG05]. Dichos sistemas persiguen el objetivo fundamental de independizar la lógica de decisión del conocimiento necesario para formalizar la solución a los problemas a los que se aplican. De esta forma, un sistema experto está compuesto por lo general de un motor de análisis o inferencia, responsable de la toma de decisiones, y de una base de conocimiento basada en la especificación de baterías de reglas, cuya elaboración es responsabilidad del experto humano.

sión, ante la evidencia de haberse detectado un determinado patrón de ataque.

En general, las reglas en las que estos sistemas codifican el conocimiento experto responden al modelo *if-then*, de forma que habitualmente están compuestas por dos elementos: un conjunto de condiciones o hechos, y un conjunto de acciones a llevar a cabo en caso de que se cumplan las condiciones correspondientes. De esta forma, cuando el motor de inferencia detecta que un determinado evento satisface el conjunto de condiciones de una regla, simplemente dispara el conjunto de acciones asociado a dicha regla, pudiendo confirmar de esta forma nuevos hechos (satisfacer nuevas condiciones) que permiten a su vez disparar nuevas reglas, de forma encadenada, hasta que es explotado todo el conocimiento disponible.

A pesar de que esta filosofía de detección resulta muy precisa, por lo general, el gran volumen de datos de auditoría que habitualmente es necesario procesar a través del motor de inferencia puede llegar a introducir serios problemas de rendimiento durante su explotación. El proyecto Haystack [S.88] fue, en 1.988, el primer caso de sistema experto aplicado a la Detección de Intrusiones, si bien han aparecido desde entonces (sobre todo en los primeros años de la evolución) otras muchas soluciones que aplican conceptos similares, como pueden ser MIDAS[1] [2] [SM88], IDES, NIDES, EMERALD, DIDS o CMDS[3] [P.96].

Los anteriormente mencionados Snort, Suricata, Bro, NFR, RealSecure, CISCO Secure IDS, o Dragon usan esta técnica, en mayor o menor medida. Sin embargo, por lo general, el conocimiento experto utilizado en estos sistemas no contempla la posibilidad de que determinados patrones o firmas habiliten el disparo encadenado de nuevas reglas, lo que representa la pérdida de uno de los pilares fundamentales de los sistemas expertos. De esta forma, la verdadera esencia de este enfoque queda muy frecuentemente sin explotar y restringida a tareas secundarias de registro de información forense, debido a la complejidad administrativa que introduce su utilización plena.

Por otro lado, este tipo de sistemas presentan por lo general una importante limitación, relacionada con su incapacidad para representar la dimensión temporal, la cual es utilizada frecuentemente para resolver la ordenación cronológica de los eventos a medida que se van produciendo en el sistema. El conjunto de todos los hechos demostrables de la base de

[1] MIDAS: Multics Intrusion Detection and Alerting System.

[2] MIDAS utiliza el denominado Production-Based Expert System Toolset (P-BEST), desarrollado inicialmente por Alan Whitehurst y evolucionado posteriormente en los desarrollos de IDES, NIDES y EMERALD [LP99].

[3] CMDS: Computer Misuse Detection System.

conocimiento no lleva asociada de forma implícita ningún tipo de información temporal, por lo que dichos sistemas son simplemente incapaces de determinar el orden en que se han ido confirmando los diferentes hechos, y por lo tanto de tomar consideración cronológica alguna durante el proceso de decisión. Aunque en algunas versiones comerciales, se disponen de *plugins* que tratan de solventar esta información en lo que se denominan *preprocesadores* orientados generalmente a ataques DoS (*Denial Of Service*). En general, aquellas problemáticas de seguridad que requieren necesariamente dichas consideraciones temporales, como pueden ser el escaneo de puertos[1] o la denegación de servicio, son resueltas mediante métodos específicos, añadidos como módulos del sistema de detección [Sno05].

Asimismo, otro inconveniente característico de estos sistemas consiste en que las decisiones del motor de inferencia están limitadas cualitativamente a la calidad del conocimiento experto de los responsables de seguridad encargados de la especificación de la base de conocimiento. Además, el modo en que dichos responsables especifican el mencionado conocimiento experto puede llegar a hacer que la base de conocimiento sea en muchos casos completamente ininteligible, debido al altísimo nivel de detalle que puede alcanzarse en la elaboración de las baterías de reglas, sobre todo en entornos en los que es necesaria una alta optimización de dichas baterías.

Por último, es interesante prestar atención al hecho de que los sistemas expertos son susceptibles de ser utilizados con el objetivo de combinar múltiples parámetros de Detección de Intrusiones, de forma que es posible construir un modelo homogéneo que, además, introduce en el análisis el concepto de certidumbre. En cualquier caso, la representación de la certidumbre en dichos sistemas de producción presenta grandes limitaciones, tal y como se desprende de los estudios realizados por Judea Pearl [J.88].

2.2.1.1.4 Diagramas de Transición de Estados

Esta propuesta pretende optimizar el volumen del flujo de alarmas que son transmitidas hacia el operador humano. En concreto, su objetivo fundamental consiste en identificar, representar y analizar las diferentes fases por las que suelen pasar, en general, los distintos tipos de agresiones contra sistemas de información. De esta forma, en lugar de utilizar un único

[1] Escaneo de Puertos o Port Scanning [Fyo97]: "An attack that sends client requests to a range of server port addresses on a host, with the goal of finding an active port and exploiting a known vulnerability of that service."[R.00]

patrón para representar un determinado ataque, este tipo de sistemas utiliza diagramas de transición de estados, o autómatas finitos[1][WF06], que representan los distintos estados que debe alcanzar un intruso potencial para llevar a cabo con éxito dicho ataque. Asimismo, las transiciones entre estados se caracterizan mediante condiciones, o asertos, que deben ser satisfechas para que se materialicen dichas transiciones. Cada una de estas transiciones equivale a una regla de los anteriores sistemas expertos. Por otro lado, el estado inicial de cada uno de los diagramas representa el estado del sistema antes de ser agredido, mientras que el último de los estados se caracteriza por representar el éxito del ataque. Lógicamente, el sistema a proteger no debería llegar a ninguno de los múltiples estados finales bajo ningún concepto.

De este modo, mediante el seguimiento del estado de avance en el que se encuentra un determinado atacante, es posible liberar al administrador humano de una gran cantidad de notificaciones de alarma. Solamente se le informa de aquellos eventos que pueden llevar al sistema a un estado inseguro. La primera aproximación de Sistema de Detección de Intrusiones basado en diagramas de transición de estados fue STAT [VG00, IK95, yKR92] , que posteriormente fue portado a sistemas Unix bajo el nombre de USTAT[K.93]. Las figuras 2.3 y 2.4, extraídas de los estudios de Steve Eckmann, describen de forma gráfica un escenario de ataque contra servidores FTP, escrito en lenguaje STATL[EVK00].

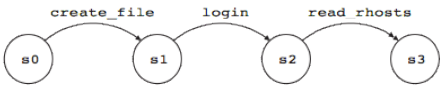


Figura 2.3: Escenario STAT para detección de ataques FTP.

Sin embargo, a pesar de lo aparentemente innovador de esta propuesta, la aportación conceptual que provee con respecto a los anteriores sistemas de detección basados en sistemas expertos no es muy grande. Ambos modelos son semánticamente equivalentes [Eck01], por lo que cualquier diagrama de transición de estados puede representarse mediante un conjunto de reglas de producción, y viceversa. Eckmann incluso propone la traducción automática[2] de baterías de reglas de Snort[3] [R+99] en escenarios de

[1] Denominados escenarios.

[2] Para lo cual, el propio Eckmann ha desarrollado la herramienta Snort2Statl.

[3] A pesar de que en la mayoría de los casos solamente se utiliza la capacidad de Snort de identificar patrones estáticos, es posible utilizar toda la potencia conceptual de los sis-

```
use bsm, unix;
scenario ftp_write
{
  int user;
  int pid;
  int inode;

  initial state s0 { }

  transition create_file (s0 -> s1)
    nonconsuming
  {
    [WRITE w] : (w.euid != 0) && (w.owner != w.ruid)
    { inode = w.inode; }
  }

  state s1 { }

  transition login (s1 -> s2)
    nonconsuming
  {
    [EXECUTE e] : match_name(e.objname, "login")
    {
      user = e.ruid;
      pid = e.pid;
    }
  }

  state s2 { }

  transition read_rhosts (s2 -> s3)
    consuming
  {
    [READ r] : (r.pid == pid) && (r.inode == inode)
  }

  state s3
  {
    {
       string username;
       userid2name(user, username);
       log("remote user %s gained local access", username);
    }
  }
}
```

Figura 2.4: STATL del escenario STAT para detección de ataques FTP.

STAT [yKR92]. Además, se da la circunstancia de que, en muchos casos, los requisitos de rendimiento obligan a utilizar reglas muy ligeras y con pocas relaciones entre ellas, por lo que dichos modelos son muy poco utilizados en la práctica.

2.2.1.1.5 Redes de Petri

Una variante interesante de los sistemas de detección basados en diagramas de transición de estados, es la propuesta en 1.994 por Sandeep Kumar y Eugene Spafford, en su sistema IDIOT[1] [ySE94b], el cual hace uso de las de-

temas de producción, a través de dos parámetros especiales, denominados activates y dynamic [Sno05].

[1]IDIOT: *Intrusión Detection In Our Time*. Los autores deseaban hacer referencia explícita, no exenta de cierta ironía, al habitual desfase de tiempo que existe desde que se construye el prototipo experimental hasta que se lanza el desarrollo del sistema de pro-

nominadas Redes de Petri [CC98] para representar escenarios de intrusión. Dichas Redes de Petri se caracterizan por poseer una capacidad semántica superior a los tradicionales autómatas finitos [M.03a], y su uso está muy extendido, sobre todo, en sistemas de producción automática. En concreto, una de las características más potentes que presentan es su capacidad de representar procesos o tareas concurrentes, de forma que un determinado sistema puede encontrarse en un momento dado, no en un único estado de ejecución, sino en varios. Para ello, una Red de Petri está compuesta, no sólo por un conjunto de estados y un conjunto de transiciones entre dichos estados (como era el caso de los diagramas de transición de estados, referenciados anteriormente), sino además por un conjunto de *tokens*, marcas o testigos, que habilitan el disparo de dicho conjunto de transiciones. De esta forma, una determinada transición no se materializará realmente, incluso aunque se confirme su condición asociada, a menos que exista un número adecuado de tokens en los estados origen de dicha transición (que pueden ser varios, al contrario que en el caso anterior) que la habiliten. Por otro lado, una vez habilitada y disparada dicha transición, se consumen los tokens ya utilizados en los estados origen, y se crean nuevos tokens en los estados destino de la misma.

Sin embargo, a pesar de la mayor potencia de representatividad de las Redes de Petri, la solución propuesta en el proyecto IDIOT, dada la complejidad administrativa que requiere la elaboración de redes que aprovechen todo el potencial del modelo general, no hace uso de toda la capacidad semántica que tiene a su disposición (sobre todo en lo relativo a la mencionada especificación de concurrencia) [ySE94b], por lo que es posible construir un modelo equivalente basado en sistemas expertos. Además, al igual que en los casos anteriores, la calidad de la base de conocimiento sigue dependiendo de la calidad del conocimiento experto del administrador humano responsable de la configuración del sistema de detección.

2.2.1.1.6 Detección basada en Modelos

Esta técnica fue propuesta por Garvey y Lunt en 1991[yLT91] e introduce el concepto de modelo de ataque, en combinación con un mecanismo de razonamiento basado en evidencias que tiene como objetivo la inferencia de conclusiones. Mantiene ciertas similitudes con los métodos basados en diagramas de transición de estados, los cuales son en este caso denomina-

ducción.

dos modelos, así como con los métodos basados en probabilidad condicional. De esta forma, la omnipresente base de conocimiento está compuesta en esta ocasión por un conjunto de modelos, cada uno de los cuales representa una secuencia de comportamientos constitutivos de un determinado ataque. Posteriormente, durante la explotación del sistema, a partir de las acciones de cada uno de los usuarios del mismo, el motor de inferencia va descartando probabilísticamente ciertos modelos, de forma que el conjunto de los ataques posibles que pueden corresponderse con la situación actual de dicho sistema se va reduciendo progresivamente. De esta manera, si se da el caso de que dicho conjunto de modelos potenciales queda vacío, se puede concluir que el comportamiento del usuario en cuestión es legítimo. Además, este método va más allá del mero análisis estadístico ya que anticipa, antes de llevar a cabo cada una de las comprobaciones, una selección de los parámetros de comportamiento a verificar, en base al conjunto de modelos que en un momento dado aún tiene posibilidades. De esta forma, no sólo se lleva a cabo el análisis pertinente, sino que se busca continuamente cómo realizar dicho análisis de la forma más eficiente posible.

En general, el proceso completo de análisis queda reducido de este modo a dos etapas: una primera etapa en la cual se establece un conjunto de hipótesis de comportamiento, y una segunda en la cual se contrastan dichas hipótesis con la actividad registrada en el histórico de auditoría. En términos probabilísticos, si se desea obtener una verificación precisa y eficiente, la relación entre las hipótesis de comportamiento y la actividad que realmente se lleva a cabo en el sistema debe constituir el valor más alto posible [S.95] (ecuación 2.4).

$$\frac{P(\frac{Actividad}{Comportamiento})}{P(\frac{Actividad}{Cualquier\ otro\ comportamiento})} \tag{2.4}$$

Al igual que el caso de los Sistemas de Detección de Intrusiones basados en probabilidad condicional, el registro histórico de las evidencias que se producen en el sistema permite obtener un cálculo adaptativo de la probabilidad de los diferentes modelos. A medida que el sistema evoluciona, algunos de esos modelos ganan peso específico (se hacen más probables, dados ciertos comportamientos), mientras que otros pierden relevancia.

De esta forma, es posible obtener una adecuada representación del concepto de certidumbre, al igual que en el caso de los sistemas basados en probabilidades condicionales, de forma que se pueden superar ciertas limitaciones inherentes a los sistemas expertos, como pueden ser la propa-

gación de la certidumbre o la aparición de hechos contradictorios. Además, dada su capacidad para anticipar los parámetros de detección a analizar, si se configuran los distintos modelos de forma que utilicen progresivamente parámetros cada vez más detallados y complejos, es posible obtener unos requisitos computacionales muy bajos. Así pues, en primer lugar, cuando aún se tiene un gran conjunto de modelos a verificar, se analizarían parámetros de grano grueso, y posteriormente, cuando el número de modelos se va haciendo cada vez menor, se irían analizando parámetros con un mayor nivel de detalle. Por supuesto, todas estas consideraciones hacen que la Detección de Intrusiones basada en Modelos presente unos requisitos de administración extraordinarios, de forma que en la práctica su utilización es muy escasa.

Por último, es importante prestar especial atención al hecho de que este tipo de sistemas de detección no sustituyen a los Sistemas de Detección de Intrusiones basados en Detección de Anomalías, dada la especificación manual de las diferentes relaciones entre los múltiples parámetros que constituyen las colecciones de modelos. Esta técnica debe considerarse solamente como un complemento a dichos sistemas, como se explica en la obra de Pearl [J.88].

2.2.1.2 **Detección de Anomalías**

La Detección de Anomalías hace referencia al método de Detección de Intrusiones que aboga por comparar el conocimiento relativo al comportamiento habitual del sistema que se desea proteger con la actividad cotidiana del mismo. De esta forma, la operatoria del sistema de detección se basa en la elaboración y mantenimiento de un perfil de actividad normal, compuesto por un conjunto de parámetros o métricas de detección, y en el posterior contraste de dicho perfil con las acciones que se estén llevando a cabo en un momento dado. Toda acción que se desvíe significativamente del perfil habitual será inmediatamente catalogada como subversiva y provocará las acciones de respuesta correspondientes.

De esta manera, dado que la principal fuente de conocimiento del sistema de detección es la realidad cotidiana del sistema a proteger, y no el conocimiento experto proveniente del administrador humano, es posible responder ante situaciones para las cuales no existe un modelo de ataque conocido y documentado dentro de la base de conocimiento, con lo que se consigue superar ampliamente las limitaciones características de los sistemas de detección basados en Detección de Usos Indebidos. No se utiliza la

política de lista negra inherente a estos sistemas, sino que se opta por una política de lista blanca construida a partir de la observación minuciosa del sistema objetivo; de forma que es posible responder ante los *Zero-Day Attacks* mencionados en apartados anteriores, no en base a lo que se conoce como malicioso, sino a partir de lo que se sabe que es legítimo, o normal. En general, estas características propias de los sistemas de detección basados en Detección de Anomalías hacen que dichos sistemas sean calificados de completos [DDW99], si bien adolecen de cierta falta de precisión que da origen habitualmente a mayores tasas de falsos positivos.

De esta forma, con el objetivo de proporcionar una visión general sobre el funcionamiento interno de este tipo de sistemas de detección, que permita extraer las conclusiones necesarias para identificar las problemáticas existentes y enunciar los grandes retos que aún presenta esta tecnología, se describen a continuación los principios básicos de funcionamiento sobre los cuales está construida la inmensa mayoría de ellos.

2.2.1.2.1 Modelos estadísticos

Dorothy Denning describió en su histórico artículo An Intrusion Detection Model[Den87] cinco modelos estadísticos en base a los cuales es posible categorizar los diferentes parámetros o métricas de detección que alimentan actualmente a la práctica totalidad de Sistemas de Detección de Intrusiones basados en Detección de Anomalías. En concreto, el objetivo fundamental del estudio desarrollado por Denning consistía en, dado un parámetro x y una secuencia de n observaciones $x_1, ..., x_n$, determinar si una nueva observación x_{n+1}es normal respecto del conjunto de observaciones previas o si, por el contrario, se desvía de forma significativa de las mismas. Para ello, Denning propone los modelos estadísticos que se describen a continuación, algunos de los cuales fueron posteriormente incorporados dentro del proyecto IDES [LT90, JV91].

- Modelo Operacional

Este modelo está basado en la hipótesis de que la normalidad de los parámetros que componen un determinado evento puede decidirse mediante la comparación de los mismos con un conjunto de valores límite que se establecen manualmente en base al histórico de evidencias. De esta forma, aunque las observaciones anteriores no se usan directamente en la decisión, sirven para establecer esos límites particulares de las diferentes métricas de

detección. A este respecto, el ejemplo de modelo operacional por antonomasia es el de un contador de intentos de login por unidad de tiempo, a partir del cual una observación por encima del límite habitual sugiere la existencia de un intento intrusión.

- Modelo basado en media y desviación típica

Este modelo pretende generalizar el modelo operacional, aportando una mayor capacidad representativa que se consigue a partir de los conceptos estadísticos clásicos de media, varianza y desviación típica [H.04a]. Para ello, el presente modelo se basa en la hipótesis de que los valores más representativos del conjunto de las n observaciones pasadas son precisamente la media y la desviación típica, las cuales se calculan mediante las ecuaciones 2.5, 2.6, 2.7, 2.8 y 2.9.

$$Sumatorio = x_1 + ... + x_n \tag{2.5}$$

$$Sumatorio\ de\ cuaodrados = x_1^2 + ... + x_n^2 \tag{2.6}$$

$$Media = \frac{Sumatorio}{n} \tag{2.7}$$

$$Varianza = \frac{Sumatorio\ de\ cuadrados}{(n-1)} - Media^2 \tag{2.8}$$

$$Desviacion\ tipica = \sqrt{Varianza} \tag{2.9}$$

De esta forma, una nueva observación es catalogada como anómala si cae fuera de un intervalo de confianza centrado en el valor de la media y con un margen definido por el administrador como una fracción k de la desviación típica. La expresión define formalmente los límites de dicho intervalo, ecuación 2.10

$$Media \pm k \cdot Desviacion\ tipica \tag{2.10}$$

Este modelo puede utilizarse sobre distintos tipos de parámetros, como pueden ser contadores de eventos, temporizadores, y demás métricas acumulables. Además, presenta dos grandes ventajas con respecto al modelo anterior, ya que, por un lado, no necesita conocimiento acumulado sobre la actividad del sistema para definir los diferentes límites. Por el contrario, el propio modelo es capaz de aprender por sí mismo qué valores constituyen la actividad normal del sistema, y en qué intervalos de confianza debe ubicarse el comportamiento de los usuarios. Además, por otro lado,

como los intervalos de confianza pueden construirse de forma específica para cada uno de los usuarios, el modelo soporta aquellas circunstancias en las que lo que es normal para un determinado usuario sea anómalo para otro. Por último, con el objetivo de minimizar las posibilidades de sufrir Session Creeping, es posible plantear una variante de este modelo que penalice progresivamente las observaciones más antiguas, en beneficio de las más recientes.

- Modelo Multivariable

Este modelo pretende generalizar el modelo basado en media y desviación típica, analizando las potenciales correlaciones existentes entre dos o más métricas. Esta filosofía está basada en la hipótesis de que es posible obtener una mayor capacidad de discriminación a partir de combinaciones de múltiples parámetros de detección, que a partir de métricas individuales. De esta forma, por ejemplo, la decisión de clasificar el comportamiento de un determinado usuario, se realizaría en base al análisis conjunto de una colección de métricas como pueden ser tiempo de procesador consumido, número de procesos en ejecución, número de operaciones de entrada-salida solicitadas, frecuencia de acceso al sistema, duración de la sesión de usuario, etcétera. .

- Modelo basado en Cadenas de Markov

Este modelo pretende complementar a los modelos anteriores, introduciendo el concepto de estado en el proceso de análisis. Los distintos tipos de análisis descritos hasta el momento toman sus decisiones sin tener en consideración las múltiples circunstancias por las que ha podido pasar el sistema que se pretende proteger, por lo que su capacidad de representación se encuentra limitada. Por el contrario, en este caso, se propone la utilización de un mecanismo matemático orientado precisamente a resolver este problema de falta de memoria, y que es conocido habitualmente como Cadena de Markov.

Para ello, este modelo considera no sólo el evento que acaba de producirse en este instante en el sistema, sino también el que se produjo en el instante anterior. De esta forma, internamente, se construye una matriz de estado que caracteriza las frecuencias de transición entre los múltiples estados por los que puede pasar el sistema a lo largo de su ciclo de vida (en lugar de las frecuencias individuales de cada estado, como es característico

de los casos anteriores). Así pues, una determinada observación es categorizada como anómala si su probabilidad de ocurrencia, dado el estado anterior, es demasiado baja.

Por otro lado, es interesante señalar la posibilidad de que este tipo de análisis puede generalizarse aún más, con el objetivo de tomar en consideración no ya el estado inmediatamente anterior, sino un conjunto de n estados anteriores, mediante la utilización de Cadenas de Markov Generalizadas [D.97].

Por último, es importante prestar especial atención al hecho de que este tipo de análisis, que constituye un subconjunto semántico de las anteriormente mencionadas Redes Bayesianas [K.01a, CGH97, DdIA98, Cha93, RJ86], también se incorpora, de forma implícita, a la solución propuesta en el presente estudio.

- Modelo Temporal

Este último modelo pretende complementar las propuestas anteriores, analizando, además del orden en que se producen los eventos en el sistema (tal y como es característico del caso anterior), la longitud de los intervalos de tiempo existentes entre dichos eventos. Básicamente, se reduce a la aplicación del modelo basado en media y desviación típica a una métrica especial que no es más que el intervalo de tiempo transcurrido entre un determinado evento y el anterior. De esta manera, una determinada observación será clasificada como anómala si su probabilidad de ocurrir en ese preciso instante es demasiado baja. Este tipo de análisis presenta la ventaja de ser capaz de estimar tendencias de comportamiento a lo largo del tiempo, lo que le permite detectar cambios graduales en dicho comportamiento y adaptarse a la nueva realidad del sistema que se pretende proteger. También esta cuestión es tomada en consideración en el presente trabajo de investigación, con lo que el exhaustivo modelo estadístico de Denning es abordado en toda su extensión.

2.2.1.2.2 Detección de Umbral

La detección de umbral constituye la forma más simple de Detección de Anomalías. Por lo general, el principio básico de funcionamiento consiste en analizar, dentro de un intervalo de tiempo, una determinada métrica de detección, como puede ser, por ejemplo, el número de intentos de conexión fallidos por unidad de tiempo. Si una vez realizada la observación,

dicha métrica excede un determinado umbral, establecido previamente, se considera dicha observación como anómala y se dispara la acción de respuesta correspondiente [AAMP03]. Un Sistema de Detección de Intrusiones que goza de gran popularidad y que utiliza esta técnica es Tripwire [Tri06, Inc06b, ySE94a] vigente actualmente y muy activo en conferencias de seguridad internacionales (BlakcHat 2015). Este tipo de detección se corresponde fundamentalmente con el modelo operacional propuesto por Denning.

2.2.1.2.3 Detección basada en Perfiles

La detección basada en perfiles pretende generalizar el método anterior de detección de umbral, de forma que en lugar de considerarse una única métrica de detección, son analizados perfiles compuestos por múltiples parámetros que sirven para obtener una representación más precisa de la actividad que se lleva a cabo en el sistema. La siguiente tabla 2.1 incluye algunas de las métricas utilizadas por el sistema IDES [JV91, LT90].

Tabla 2.1: Métricas utilizadas en el sistema IDES.

	Métrica
1	Número de registros de auditoría generados por un usuario y por unidad de tiempo.
2	Distribución relativa de acceso a archivos y de actividad de entrada-salida por usuario.
3	Frecuencia relativa de accesos al sistema por localización física.
4	Utilización total y relativa de servicios de aplicación.
5	Utilización total y relativa de comandos de sistema.
6	Utilización de procesador y memoria por usuario.
7	Número de archivos accedidos por nivel de confidencialidad.

De esta forma, la combinación de este conjunto de métricas para un determinado usuario, constituye en cada momento un perfil actual de comportamiento, que es comparado periódicamente con un perfil almacenado que contiene los múltiples valores umbrales de dicho conjunto de métricas. Si se produce una situación en la que el perfil actual se desvía significativamente del almacenado, se considera dicho perfil actual como anómalo y se genera la respuesta pertinente [AAMP03]. En algunos sistemas de detección, además, el mencionado perfil almacenado se actualiza regularmente

con los datos contenidos en el perfil actual, con lo que se consigue la adaptación automática a potenciales cambios en el comportamiento de los usuarios [S.95]. Esta modalidad de detección se corresponde fundamentalmente con el modelo operacional propuesto por Denning, si bien, en función del tipo de comparación que se implemente internamente, puede presentar además ciertas peculiaridades propias del modelo basado en media y desviación típica, del modelo multivariable, y del modelo temporal.

2.2.1.2.4 Métricas Estadísticas

En la línea del modelo operacional, del modelo basado en media y desviación típica, del modelo multivariable y del modelo temporal propuestos por Denning, la detección basada en métricas estadísticas aboga por el análisis conjunto de los distintos parámetros de detección en base a una expresión heurística compuesta que combine las distintas componentes de anomalía posibles. Sobre este principio básico de funcionamiento está basado, por ejemplo, el sistema de detección NIDES [AFTV94], en el cual se utilizan además los conceptos anteriormente mencionados de perfil actual y perfil almacenado. En concreto, el perfil almacenado que se utiliza está compuesto, en general, por un conjunto de i métricas $s_1...s_i$, mientras que el perfil actual está compuesto por i observaciones $c_1...c_i$. Por otro lado, los valores de anormalidad o anomalía ai se obtienen como la diferencia en valor absoluto entre ambos perfiles, de forma que se obtiene un conjunto de valores $a_1...a_i$, para los que $a_i = |s_i - c_i|$. Además, no es habitual que todas las métricas tengan la misma relevancia, de forma que es necesario añadir un peso específico w_i a cada métrica. Por último, tampoco es habitual que todos los valores de anomalía tengan la misma repercusión en la decisión final. Las anomalías importantes deben tener una mayor relevancia que otras anomalías menores, por lo que es necesario considerar dichos valores al cuadrado, por lo menos [AAMP03, S.95]. Todas estas consideraciones sirven para construir la ecuación 2.11, la cual constituye el núcleo fundamental del mencionado proyecto NIDES.

$$Anomalia = w_1 a_1^2 + w_2 a_2^2 + ... + w_i a_i^2, \forall i > 0 \tag{2.11}$$

De esta forma, cualquier evento que produzca un valor de anomalía por encima de un determinado umbral, establecido de antemano, será considerado como anómalo y suscitará la respuesta correspondiente por parte del sistema de detección. En general, este tipo de sistemas de detección (al

igual que el caso anterior, de detección basada en perfiles) permite advertir utilizaciones anómalas de los recursos computacionales del sistema que se desea proteger de forma eficaz, si bien presenta ciertos inconvenientes de importancia, como pueden ser su incapacidad para representar información temporal (al igual que el caso de la Detección de Usos Indebidos basada en sistemas expertos), la posibilidad de sufrir ataques de *Session Creeping*, y su falta de precisión en entornos en los que el comportamiento de los usuarios es ligeramente errático o voluble.

2.2.1.2.5 Sistemas basados en Reglas

La Detección de Anomalías basada en sistemas de reglas pretende aportar una mayor capacidad semántica a los modelos anteriores. Mientras los sistemas de detección basados en métricas estadísticas resuelven el problema de la comparación de perfiles mediante formulación matemática, sus homólogos basados en sistemas de reglas proponen, como se describe en apartados anteriores, la utilización de un motor de inferencia responsable de interpretar baterías de reglas que formalizan conocimiento experto. De esta forma, dado que los mencionados sistemas basados en reglas presentan un mayor poder expresivo [AAMP03], se pretende mediante su utilización la resolución de algunos de los problemas inherentes a dichos modelos anteriores, como son el modelado explícito de información temporal, y la adaptación del sistema de detección al comportamiento anárquico de algunas tipologías de usuarios.

Un sistema de Detección de Anomalías basado en sistemas de reglas es ADS[1] [yKS97]. Dicho sistema de detección está orientado a la máquina y es capaz de analizar información del Sistema Operativo proveniente de los niveles de servicio, o de llamada al sistema, y de interpretación de comandos. Para ello, es necesario especificar conjuntos de reglas que representan el comportamiento de cada uno de los usuarios del sistema, incluyendo métricas como el tipo de llamadas al sistema permitidas, número de solicitudes de servicio realizadas, etcétera. Una vez elaborado ese perfil de comportamiento, durante la explotación del sistema de detección, cualquier actividad que sobrepase los límites establecidos en el mismo será considerada como anómala por el motor de inferencia, y se ejecutarán las acciones de respuesta correspondientes.

En general, esta técnica está basada en la hipótesis de que las secuencias

[1] ADS: Anomaly Detection System.

de eventos no se producen aleatoriamente, sino que por el contrario siguen patrones regulares e identificables[1]. Por ello, un objetivo fundamental que se aborda es el del análisis de anomalías en la ordenación de eventos, lo que hace que la Detección de Intrusiones adquiera una mayor calidad. Otros ejemplos de sistemas de detección que utilizan esta técnica son Wisdom and Sense[2] [yVH92] y TIM[3] [TH90].

2.2.1.2.6 Redes Neuronales

Dado que el objetivo fundamental de todo Sistema de Detección de Intrusiones basado en Detección de Anomalías consiste en deducir, a partir de un conjunto de $x_1 ... x_n$observaciones, el grado de normalidad del evento $x_n + 1$[Den87], la aplicación del mecanismo matemático de aprendizaje e inferencia conocido como Red Neuronal se revela como una opción natural, con prometedoras posibilidades [yNP02, P.92]. En concreto, el objetivo fundamental que se persigue con este enfoque es el de representar todo el conocimiento relativo a los perfiles de comportamiento de los usuarios de una forma implícita al mecanismo de inferencia, y no mediante los habituales esquemas explícitos inherentes a las aproximaciones basadas en perfiles, métricas estadísticas o sistemas expertos. De esta forma, se enriquece, potencialmente, el conocimiento relativo a las múltiples relaciones existentes entre los diferentes parámetros de detección, con lo que la calidad de las decisiones inferidas es teóricamente superior [AAMP03, S.95].

Para ello, se organiza un esquema de funcionamiento compuesto por dos fases: aprendizaje, o entrenamiento, y explotación. De esta manera, en primer lugar se entrena la red neuronal con un histórico de la actividad de que se ha registrado en el pasado, el cual, además de las métricas propias del perfil de comportamiento, debe incluir un parámetro de detección adicional que representa la clasificación de dicha actividad. Dicho parámetro de clasificación representa el hecho de si la actividad analizada ha supuesto un ataque (y en caso afirmativo, el tipo de ataque) o no contra la seguridad del sistema, de forma que su finalidad última no es otra que la de etiquetar el conocimiento disponible para así proporcionar a la posterior fase de explotación conciencia sobre el hecho clasificatorio.

[1] Esta técnica también es conocida habitualmente como Generación de patrones predictivos o Predictive Pattern Generation [S.95].

[2] Wisdom and Sense: Sabiduría y Sensibilidad.

[3] TIM: Time-based Inductive Machine.

Posteriormente, una vez que la red neuronal ha sido entrenada convenientemente, tanto el conjunto de los múltiples perfiles de comportamiento, como sus métricas componentes, quedan representados implícitamente y de forma unificada en las múltiples conexiones neuronales que componen la citada red. De esta manera, la red en cuestión puede ser finalmente utilizada para obtener conclusiones acerca del tipo de categorización que debe aplicarse a las nuevas observaciones que se registren en el sistema. Si los parámetros de detección que componen un determinado evento mantienen cierta similitud con los recogidos por la red neuronal, dicho evento será considerado anómalo y el sistema de detección responderá en consecuencia. De esta forma, estos sistemas de detección presentan poderosas posibilidades: son capaces de adaptarse por sí mismos a sistemas computacionales cuya realidad evoluciona a lo largo del tiempo, mantienen una alta estabilidad del conocimiento asimilado (incluso en situaciones sujetas a la influencia de componentes de ruido), son capaces de inferir conclusiones de alto nivel de abstracción, sin necesidad de hipótesis estadísticas previas, y además son muy eficientes, una vez puestos en explotación.

Sin embargo, la utilización de redes neuronales introduce un serio inconveniente en lo relativo a la inferencia de relaciones de tipo causa-efecto, ya que dentro de su naturaleza existe una absoluta incapacidad de explicación de las conclusiones que ofrecen. La suma de sus múltiples conexiones, una vez entrenadas adecuadamente, proporciona unos resultados precisos y de confianza, pero es imposible determinar cuáles son las causas exactas que producen dichos resultados[CGH97]. Por otro lado, la mencionada fase de aprendizaje requiere por lo general unos costosos esfuerzos administrativos de supervisión, que hacen que, por lo general, este tipo de sistemas de detección vea reducida su aplicación a condiciones de laboratorio [GILB00, GA99, CM98, yLM98]. A pesar de tratarse de un enfoque especialmente particular, puede clasificarse de una forma intuitiva dentro del modelo multivariable de Denning. En la actualidad los conceptos se mantienen y siguen siendo usadas para la detección de anomalías: [AKBD15], [WPL+15], [AJA15], etcétera, ya que es un campo muy prolífico.

2.2.1.2.7 Lenguajes de Especificicación de Intenciones

Un método que aboga por la especificación apriorística del comportamiento de los usuarios del sistema, es la utilización de lenguajes interpre-

tados a través de los cuales especificar dicho comportamiento[1]. De esta forma, es posible la verificación en tiempo real del grado de desviación que está sufriendo el comportamiento actual de un determinado usuario, con respecto a su comportamiento ideal, especificado mediante uno de dichos lenguajes. Éste es un método que, además de permitir la especificación de comportamiento legítimo, también puede utilizarse para representar conocimiento experto sobre comportamientos maliciosos de los que el administrador humano tenga constancia, debido al alto grado de capacidad expresiva que presenta.

Algunos casos de éxito de Sistemas de Detección de Intrusiones que utilizan esta técnica son IDML[2] [LY01], NFR o Stalker[S.96]. En todos ellos, se utilizan lenguajes de especificación expresamente desarrollados para cada caso, si bien también existen soluciones de detección que utilizan lenguajes de propósito general de construcción de sistemas expertos, aplicados a la especificación de escenarios de Detección de Intrusiones, como pueden ser P-BEST[3] o CLIPS[4][RG05b].

2.2.1.2.8 Clasificación Bayesiana

Los métodos de clasificación Bayesiana [J.88] están basados en los conceptos clásicos de Teoría de la Probabilidad conjunta [H.04a] y probabilidad condicional, y se caracterizan fundamentalmente por conseguir la clasificación no supervisada de objetos. Esta propiedad de la no supervisión permite que estos métodos de análisis presenten unos requisitos administrativos mínimos y que los hacen especialmente atractivos, ya que elimina la necesidad de que sus inherentes procesos de aprendizaje deban ser administrados por el operador humano, como es menester, por ejemplo, en los casos de los métodos de detección de umbral, o en los basados en perfiles, en métricas estadísticas, en sistemas de reglas, o en redes neuronales [S.95].

De este modo se consigue una adaptación natural e instantánea a la evolución a lo largo del tiempo de los sistemas que se pretende proteger, ya que es posible mantener, en paralelo con los procesos de inferencia y decisión, perennes procesos de aprendizaje automático. Además, un sistema de clasificación Bayesiano permite la identificación automática del número de

[1] Conocidos como lenguajes de especificación de intenciones o Intent Specification Languajes o ISL [AAMP03].

[2] IDML: Intrusion Detection Markup Languaje.

[3] Utilizado dentro del proyecto EMERALD, entre otros.

[4] Utilizado dentro de los proyectos CLIPSNIDS [ACN03] y DIDS, entre otros.

categorías de clasificación, dado el conjunto de datos a clasificar, permite operar en base a datos incompletos, permite el análisis unificado de parámetros, sin los habituales criterios y restricciones de excepción presentes en la mayoría de sistemas de detección, e incluso soporta el análisis homogéneo de parámetros discretos y continuos; parámetros que, además, pueden ser tanto cuantitativos como temporales [VP01].

Sin embargo, a pesar de sus enormes posibilidades, también presenta ligeros inconvenientes. Por un lado, estos métodos aún presentan cierta dificultad para definir los umbrales de detección más adecuados; tarea que sigue siendo responsabilidad del administrador de seguridad. Por otro lado, como es característico de cualquier sistema de adaptación automática al sistema objetivo, presentan la teórica posibilidad de sufrir ataques de *Session Creeping* [CGH97]. Algunas propuestas de Sistemas de Detección de Intrusiones que utilizan métodos de clasificación Bayesianos son los proyectos SPAMBayes [yWB04], ADAM y eBayes[1], así como los estudios realizados por Steven Scott [S.04] , Nasser Abouzakhar et al. [AGMG03], Christopher Krügel et al. [KC03b], Burroughs et al. [BWC02], y Nong Ye y Mingming Xu [M.00]. La figura anterior ilustra un sistema de clasificación bayesiano utilizado para Detección de Intrusiones orientada a la máquina [S.95].

2.2.1.2.9 Algoritmos Genéticos

Los denominados Algoritmos Genéticos [D.02] constituyen una técnica de búsqueda y optimización, perteneciente al área de conocimiento de los denominados Algoritmos Evolutivos [ES03], que también ha sido propuesta dentro del área de conocimiento de la Detección de Anomalías, en concreto como componente fundamental del motor de análisis del proyecto GASSATA[2] [L.98], desarrollado por Ludovic Mé. En general, este tipo de métodos están basados en la mecánica de la Genética y de la Selección Natural [S.02]. De esta forma, el proceso de búsqueda u optimización se organiza en base a los principios genéticos clásicos de herencia, mutación, selección y recombinación, para a partir de cromosomas, formados por vectores de parámetros de detección, dar origen a un conjunto de generaciones sucesivas cuyos últimos miembros se ajusten perfectamente al problema propuesto. Al finalizar dicho proceso evolutivo, sólo habrán sobrevivido

[1] Proyecto desarrollado por SRI International, como componente del proyecto EMERALD.

[2] GASSATA: A Genetic Algorithm as an Alternative Tool for Security Audit Trails Analysis.

los más fuertes. De este modo, los resultados obtenidos por Mé fueron realmente prometedores, presentando una tasa de acierto del 0,996 y una tasa de falsos positivos del 0,0044. Además, el tiempo necesario para obtener dicha solución fue mínimo, por lo que la sobrecarga introducida en el sistema se mantuvo en unos niveles realmente bajos. Otros estudios interesantes desarrollados en esta línea pueden ser los llevados a cabo por Dong Kim et al. [KD05], por Wei Li[W.04], por el grupo de investigación de Jan Eloff [PM04], por Susan Bridges y Rayford Vaughn [BV00b], por Sinclair et al. [SC99], y por Mark Crosbie y Gene Spafford [CS95]. Otros ejemplos más actuales: [PSMP15], [SVK15], etc.

2.2.1.2.10 Lógica Difusa

La denominada Lógica Difusa[1] [Cox94] es una extensión de la Lógica de Boole que se caracteriza por ser capaz de representar el concepto de certidumbre, o grado de veracidad, aproximándose de esa forma al concepto idóneo de Probabilidad [CGH97, DdIA98]. De este modo, mientras la lógica clásica utiliza únicamente los conceptos absolutos de verdadero o falso, la lógica difusa utiliza múltiples niveles o grados de veracidad, proporcionando de esa manera una mayor aproximación al razonamiento humano. Por ejemplo, un Sistema de Detección de Intrusiones basado en lógica difusa puede representar el concepto muy peligroso. Por ello, es posible que un determinado evento sea clasificado como perteneciente (con diferentes grados de pertenencia) a más de una clase simultáneamente [BV00b].

Por otro lado, es interesante tomar el consideración el hecho de que esta tecnología es susceptible de ser utilizada por sí sola, como elemento fundamental del proceso de Análisis, o bien de usarse como complemento a otras técnicas de detección, como es el caso del proyecto IIDS[2], desarrollado en la Universidad de Mississippi por [BV00a, BV00b]. Otras Susan Bridges y Rayford Vaughn[3] investigaciones desarrolladas al respecto son las llevadas a cabo por John Dickerson et al. en el proyecto FIRE[4] [DD00, DJKD01], así como los estudios realizados por Chavan et al. [CSD+04], Jonatan Gómez y Dipankar Dasgupta[yDD01], y Jianxiong Luo [J.99]. La lógica difusa aplicada al campo de la Detección de Intrusiones sigue siendo muy prolítica en

[1] Lógica Difusa o Fuzzy Logic.

[2] IIDS: Intelligent Intrusion Detection System.

[3] En el desarrollo de este proyecto, Bridges y Vaughn utilizaron Lógica Difusa combinada con Algoritmos Genéticos.

[4] FIRE: Fuzzy Intrusion Recognition Engine.

publicaciones más actuales: [RS15], [BV15], [EFB+15], [PS15], [PPWP15], etc.

2.2.1.2.11 Máquinas de Vector Soporte

Las denominadas Máquinas de Vector Soporte[1] [V.95] constituyen un método no lineal basado en optimización cuadrática [GAH05, ySA03] de aprendizaje supervisado dirigido a la resolución de problemas de clasificación y regresión. Por lo tanto, el objetivo principal de dichos métodos ante el problema de la Detección de Intrusiones se reduce a clasificar la actividad de los usuarios, en principio en las dos grandes categorías canónicas: Maldad y Bondad. Este modo de operación es similar a los que caracterizan a otros enfoques, como pueden ser las redes neuronales o los clasificadores bayesianos, de forma que su aportación conceptual es limitada. Por otro lado, la propia definición del término incluye una premisa, como es el hecho del aprendizaje supervisado, que en general supone un importante obstáculo para la eficiente utilización de un hipotético Sistema de Detección de Intrusiones que resultara de la aplicación de las referidas máquinas de vector soporte. Algunas investigaciones en las que se han aplicado los principios fundamentales de dichas máquinas de vector soporte son las llevadas a cabo por Crina Groçan et al. [GAH05], por Srinivas Mukkamala y Andrew Sung [ySA03, MS05], por Zonghua Zhang y Hong Shen [H.04b], por Heller et al. [HK03], y por Hu et al. [HW03]. Dentro de las más recientes, ya que se usa de forma extendida en la literatura se pueden encontrar: [WDHL15], [SVK15] ó [PSMP15]. Aunque generalmente se usa esta técnica en combinación con otras como Algoritmos Genéricos, o C4.5, KNN o K-means.

2.2.1.2.12 Data Mining

El denominado Data Mining[2] [TP05, HD01], también conocido como Knowledge Discovery in Databases[3], es una técnica de búsqueda automática de patrones o pautas regulares de conocimiento en ingentes volúmenes de información, para la cual se utilizan fundamentalmente técnicas estadísticas y de reconocimiento de patrones. De este modo, la esencia última de

[1] Máquinas de Vector Soporte o Support Vector Machines [SVM].

[2] Data Mining o Minería de Datos.

[3] Knowledge Discovery in Databases (KDD) o Descubrimiento de Conocimiento en Bases de Datos.

dicha aproximación no difiere sustancialmente de otras alternativas propuestas anteriormente, salvo por su orientación no supervisada y por los grandes requerimientos de eficiencia que presenta. De esta forma, se pretende dar un paso adelante en la tecnología estadística clásica mediante la automatización del proceso de extracción de conocimiento. En este apartado aparecen las citas más representativas de cada uno de los conceptos originales, aunque actualmente hay a su vez mucha literatura aplicada sobre este campo.

Por otro lado, aunque los problemas específicos a los que puede enfrentarse el Data Mining son múltiples, son tres los que predominan habitualmente, ya que proporcionan un conocimiento de mayor calidad: Clasificación, Análisis Multivariable de Relaciones y Análisis de Secuencia [WL98]. A continuación se describen someramente las mencionadas problemáticas:

- Clasificación

El objetivo fundamental del problema de clasificación consiste en asignar una categoría a cada uno de los hechos de la base de datos. Por otro lado, puede darse la circunstancia de que el conjunto de categorías esté definido de antemano, o que deba ser inferido a partir de los datos. A este respecto, las Redes Bayesianas en general, y las Redes Bayesianas de tipo Naïve en particular, constituyen el paradigma por excelencia de clasificación automática a partir de los datos [CGH97, DdIA98].

- Análisis Multivariable de Relaciones

En este caso el problema consiste en identificar las relaciones de correlación y causalidad existentes entre las múltiples variables que componen cada uno de los hechos de la base de datos, mediante la aplicación de tests estadísticos de independencia e independencia condicional. Al igual que en el caso anterior, las Redes Bayesianas suponen un método excepcional para la resolución de dicho problema [CGH97, DdIA98]. Por otro lado, es interesante prestar atención a la absoluta correspondencia existente entre este aspecto del Data Mining y el modelo multivariable propuesto por Denning.

- Análisis de Secuencia

En esta ocasión el problema consiste en identificar las relaciones de correlación y causalidad existentes entre los múltiples estados por los que pueden pasar las secuencias de eventos que tienen origen en un determinado sistema de información. En este caso, los denominados Modelos de Markov

constituyen una excelente aproximación, si bien están superados por las anteriormente mencionadas Redes Bayesianas, las cuales, aunque presentan unos mayores requisitos computacionales, proporcionan una mayor capacidad de representación [CGH97, DdIA98], imprescindible para la resolución de problemas complejos. Al igual que el caso anterior, es interesante prestar atención a la absoluta correspondencia existente entre esta cuestión y el modelo basado en Cadenas de Markov propuesto por Denning.

Son abundantes las investigaciones realizadas en el campo de la Detección de Intrusiones que aplican técnicas de Data Mining. Algunas de las más representativas son las llevadas a cabo por Guy Helmer et al. [HG03], por el equipo de investigación de la Universidad de Minnesota [LA03, DEK+02], por Wenke Lee y Salvatore Stolfo[1] [LW99, LS98], de la Universidad de Columbia, y por Christina Warrender et al. [WP99].

[1] Lee y Stolfo utilizan un enfoque de inducción de reglas a partir de los datos muy similar al utilizado en el proyecto TIM.

2.3 HIDS. Técnicas de Análisis específicas para Host Intrusion Detection Systems

Los Sistemas de Detección de Intrusiones para Sistemas Operativos, o Host, como se vienen denominando tradicionalmente requieren de datos para poder realizar su labor. Ha pasado bastante tiempo desde que se empezó a tener ideas sobre cómo facilitar la auditoría de sistemas en orden de garantizar la seguridad de los mismos. Existen incluso documentos, de hace bastante tiempo, que afirman que sin la capacidad de generar y almacenar información de auditoría, no se puede considerar un sistema como seguro [Pic78]

La auditoría permite dos puntos de acción fundamentales: el primero y más importante es que si ha acaecido una violación de la seguridad del sistema, los sucesos que se han ido produciendo son repetibles; así mismo, como segundo punto se tiene el que se va a poder predecir el comportamiento de los usuarios y actuar en consideración si se estima un posible ataque en base al conocimiento anterior.

Los Sistemas de Auditoría tienen que ser capaces de:

- Aceptar datos de los procesos que puedan registrar sus propios datos de auditoría.
- Recoger la suficiente información como para deducir lo que está pasando en el sistema.
- Recolectar datos en la totalidad del sistema a todos los efectos.
- Hacer acopio de sucesos relativos a la seguridad en otras partes interesantes, aun sin ser dependientes del host.

Otro de los problemas habituales en este tipo de sistemas es el hecho de que los eventos, evidencias o muestras se producen muy rápido. Usualmente se afronta esta tesitura en genérico como en la plataforma de [Pic78], CMW[1], Recolección Selectiva y Reducción Selectiva. La Reducción Selectiva se basa en recoger suficientes datos para cumplir los requisitos de identificación de usuarios, objetos accedidos, nivel de seguridad, etcétera. La Recolección Selectiva es idéntica al caso anterior pero recogiendo selectivamente cualquiera de las partes anteriores acorde al patrón que se esté buscando. Así

[1] Compartmented Mode Workstation

mismo, los datos deberán ser almacenados bien localmente o bien externamente en centros de recopilación de información. El hecho de almacenar la información localmente sucinta un riesgo de poder ser alterado indebidamente. La recogida de datos que plantearon en su momento abordaba 3 niveles importantes de modo que la captura fuera global: intercepción de llamadas de sistema (syscall) en kernel, en la interfaz que se le proporciona por el sistema operativo al usuario y a captura de interacción entre sistema operativo y aplicación. El problema asociado es el volumen de datos generados, dado que son muchos los recursos a monitorizar: sistema, aplicaciones, usuarios, etc.

2.3.1 **Preámbulo: Ataques Mimicry en Sistemas de Detección de Intrusiones**

Se hace un estudio [WS02] sobre diferentes tipos de sistemas de detección basados en host (HIDS) para ver la seguridad que presentan contra diferentes ataques evasivos. Los ataques denominados Mimicry permiten a un atacante realizar el ataque sin ser detectado por ninguno de los sistemas existentes. Para poder demostrar este tipo de ataques se desarrolla una aplicación marco que permitiera efectuarlos, demostrando que era viable realizarlos contra la seguridad existente en una implementación concreta de HIDS.

Los autores argumentan que para detección de anomalías, casi todos las llamadas al sistemas se pueden traducir en operaciones de no-operación (no-op), de forma que puedan desorientar al sistema de detección realizando finalmente la tarea que se desea. Hay mucha literatura que explica diferentes mecanismos de detección que son susceptibles a engaños mediante este tipo de técnica:

- Sistemas basados en el análisis de llamadas al sistema [SF94, FHSL96a, AG99, SH98].
- Minería de datos [WL98].
- Redes Neuronales [AG99].
- Autómatas Finitos [CM00].
- Modelos de Markov Ocultos [WP99].

- Reconocimiento de patrones en secuencias de comportamiento [TL97, TL99].

Una de los puntos de partida de este tipo de técnica es el hecho que no todo código malicioso tiene la necesidad de llamar a una llamada al sistema de bajo nivel. Como segunda arma los atacantes suelen tener de su mano la paciencia, esperar hasta que la secuencia sea aceptada por el sistema de detección como comportamiento normal para lanzar el ataque, incluso conduciendo el aprendizaje del sistema en uno u otro sentido. El atacante contra sistemas basados en detección de llamadas de sistema, puede introducir un troyano que se encargue de modificar las librerías a las que se ha llamado para realizar una determinada tarea.

Una parte importante que se debe tener en consideración cuando se realizan este tipo de ataques es el hecho de que generalmente, después de haber realizado el ataque, el sistema entra en una serie de trazas de syscalls a las que le ha llevado el código atacante, que generalmente no están contempladas por los sistemas de detección.

Otra aproximación bastante acertada es realizar un ataque mediante las cadenas de syscalls más comunes dentro de la aplicación objetivo. Es decir, crear de las diferentes trazas que se puedan tener, aquella que sea más común para que la desviación final sea lo menor posible. Siguiendo esta línea incluso se pueden realizar ataques o troyanos que emulen a la aplicación objetivo después de haber realizado el ataque para que el sistema parezca que se encuentra en un estado correcto.

Un fallo o carencia habitual en los sistemas de detección es la ausencia en el estudio de los parámetros. Otro defecto observado es que hay una forma de variar la secuencia del ataque mediante la inserción de operaciones que no hagan nada. Estas operaciones conocidas como, "sin operación (nop)", sirven para que la traza se mantenga acorde a lo que ha aprendido el sistema de detección, pero realmente el objetivo final es conseguir poner en marcha la vulnerabilidad de la seguridad deseada.

Continuando con lo expuesto en el párrafo anterior, siempre es posible generar las variaciones en el ataque que permitan llegar a la situación en la que el detector no va a ser capaz de indicar la anomalía. Un ejemplo puede ser tan simple como que una llamada *read()* de un fichero en memoria, puede ser equiparada en código a una *mmap()* seguida de un acceso a memoria. Por la misma regla, hay secuencias que pueden ser desordenadas desde el ataque original al necesario para evadir el sistema HIDS. De la misma forma unas pocas llamadas al sistema le dotan al atacante de un

poder especial, como por ejemplo la *fork()*. Esta última llamada es tratada por los sistemas HIDS desdoblando el sistema para seguir al proceso padre y al proceso hijo, lo que permite al atacante realizar el ataque en dos pasos diferentes dentro de distintos procesos. Además si se consigue un acceso a la llamada *execve()* se le ofrece al ataque la posibilidad de ejecutar cualquier cosa que desee en el sistema.

2.3.2 **Detección mediante Análisis Forense**

Es posible mejorar el conocimiento de lo sucedido en un sistema informático mediante el uso de técnicas forenses [JCR03] que no requieren de predecir la naturaleza del ataque, el conocimiento del atacante o de los detalles de los recursos del sistema y objetos afectados. Los principios incluyen la grabación de datos sobre el sistema operativo en su totalidad, particularmente los eventos y entornos en espacio de usuario, así como interpretando los eventos en capas de diferente abstracción según el contexto donde han sucedido. Los autores además, proponen la creación de una máquina de estados finita sobre la que los resultados puedan ser establecidos preferiblemente a ser inferidos.

El análisis forense es el proceso de comprensión, recreación y análisis de los eventos que han sucedido previamente. El logeo es la grabación de datos que puedan ser útiles en el futuro para la comprensión de eventos pasados. El proceso de auditoría de sistemas se compone de la recogida, examen y análisis los datos guardados.

El hecho de trabajar temporalmente después de haber sucedido la vulnerabilidad, les permite a los expertos conocer la forma en la que ha sido cometida. Para poder realizar este tipo de análisis hay que hacer uso de una serie de puntos clave:

- Debe ser considerado el sistema al completo.
- Los efectos de una acción pueden ser diferentes de los que se espera que sean.
- Los datos en ejecución son el único registro de qué pasó exactamente. Mientras que los escaneos de vulnerabilidades, producidos con anterioridad o posterioridad a la intrusión pueden ayudar ampliamente. Un conjunto de datos en tiempo real es la única fuente autorizada de la que se puede sacar información completamente fiable.

En este tipo de metodologías se tienen en cuenta una serie de premisas de partida:

- *Principio 1:* Tener en cuenta el sistema completamente. Esto incluye tanto el espacio de usuario, como el espacio de núcleo del Sistema Operativo, sistemas de ficheros, pila de protocolos de red y subsistemas asociados.
- *Principio 2*: No siempre está logeado lo que se presupone sobre fallos esperados, ataques y atacantes. No creer en ningún usuario y no confiar en ninguna política, dado que no se sabe de antemano lo que se encontrará.
- *Principio 3*: Tener en consideración los efectos de los eventos, no solo por ser debidos a una serie de acciones, si no por cómo esos efectos pueden alterar el contexto del entorno.
- *Principio 4*: El contexto ayuda en la interpretación y compresión del significado de un determinado evento.
- *Principio 5*: Cada acción y cada resultado debe ser procesado y presentado de forma que pueda ser analizado y comprendido en un análisis forense.

En el artículo se expone que una de las mayores faltas en esta técnica es la inexistencia de un análisis automatizado para poder realizar todas las comprobaciones. El resumen de realizar una solución genérica pasa por recoger no solo los eventos del kernel, si no también información sobre los tiempos, tamaños y localizaciones de alojamientos de información en memoria, lecturas y escrituras. Las herramientas deberán conseguir eventos de información acerca de enlaces abstractos, sobre todo de aquellos que vayan a la red o que sobrepasen el sistema de ficheros local. Se deben recoger datos de programas, funciones y nombres de variables. Mediante una correlación apropiada de esos nombres, eventos de memoria, contexto del sistema, entorno del programa, herramientas que cubran los principios anteriores, si es presentada de forma humanamente legible se podrá tener una herramienta adecuada para estas problemáticas.

El análisis de intrusiones a día de hoy, suele ser un trabajo difícil que se realiza generalmente a mano, debido a que los administradores carecen de información y herramientas para comprender fácilmente la secuencia de los pasos que ocurren en un ataque. Los creadores de la aplicación BackTracker [Eri08], después de producirse la intrusión, han dispuesto de una

serie de medidas para poder relacionar las evidencias entre sí. Hay una serie de factores que pueden indicar una dependencia entre eventos:

- Dependencias entre procesos. Esto se produce cuando un proceso afecta directamente en la ejecución de otro proceso. Las causas pueden ser: creación, compartición de memoria, envío de mensajes o señales...

- Dependencias proceso a fichero. Se trata de conseguir datos interesantes de ficheros del sistema para un proceso. También se puede cargar el fichero completamente en memoria y leer/escribir directamente ahí antes de guardarlo, con lo que el análisis forense deberá tener esta técnica en consideración.

- Dependencias entre proceso y nombre de fichero. Un ejemplo de esto puede ser cuando una aplicación carga un fichero con su configuración, si este contiene datos incorrectos se pueden producir situaciones peligrosas.

- Para realizar el trazado inverso de aplicaciones se logean los objetos y los eventos con las dependencias en tiempo de ejecución. De esta forma se puede construir un grafo que indique las relaciones entre todos los objetos que se han ido viendo en ejecución.

2.3.3 **Detección basada en análisis estático o instrumentación binaria del sistema**

En el documento [JCR03] los autores explican la creación de una herramienta, DOME, que realiza un análisis estático para identificar las localizaciones (espacio virtual de direcciones) de las llamadas al sistema que realizan los programas ejecutables. Después de obtener esas direcciones se monitorizan los programas en tiempo de ejecución y se verifica que la posición de memoria utilizada por el ejecutable es la que corresponde con el estudio del sistema anterior. Esta técnica es simple, práctica y aplicable al mundo real, pero si bien se debe indicar que es posible infectar el código de las llamadas a sistema, manteniendo intacta la posición del mapeo de memoria virtual.

Los autores sostienen que mediante el sistema DOME realizado obtienen una potente técnica para proteger el software contra código malicioso,

bien código inyectado dentro de los procesos (buffer overflows...), bien código generado automáticamente (virus polimórficos, troyanos que guardan sus datos cifrados, descifrando su código fuente y ejecutándose en el momento de arrancar...), o bien código malicioso ofuscado (virus, troyanos, gusanos... que ocultan su código de forma que sean difíciles de detectar).

Mediante esta técnica es posible entrar al tan ansiado campo para los administradores de detección de ataques nóveles, *Zero-day attacks*. Este proyecto ha centrado su estudio en windows 2000, con lo que el formato de estudio consecuentemente se ha fijado en formato de fichero portable ejecutable (formato PE).

Otro tema que plantean en el documento, es la posible sobrecarga del sistema al realizar la captura de eventos producidos, se indica que producen una sobrecarga al sistema de un 5 % al realizar las funciones de comparación para cada llamada a la API. Así mismo el sistema desarrollado esta limitado a las llamadas a funciones de Win32, dejando las llamadas más internas al sistema sin capturar.

Los autores plantean una serie de premisas:

- Cualquier código inyectado, generado dinámicamente u ofuscado se asume como malicioso.
- El código maligno interactúa con el sistema operativo.
- En la interacción con el sistema operativo, el código malicioso, usa las llamadas exportadas por Win32 (en el sistema operativo en estudio de los autores).
- Cuando un código malicioso se esconde de la detección y análisis mediante el uso de generación de código dinámico y ofuscación, se esconde a su vez el uso de las llamadas a Win32.
- Los ejecutables software que deben ser protegidos pueden ser desensamblados y se puede obtener la información sobre las API's usadas. Por lo tanto son monitorizables.

Se suele hacer uso de Detours [HB99] como herramienta de análisis para facilitarse el acceso al sistema operativo en cuestión, herramienta que se introducirá y comentará en el siguiente capítulo.

La investigación últimamente esta decantándose en lo que se denomina instrumentación binaria dinámica, utilizada comúnmente en los estudios para realizar Análisis Forenses, con origen en el paper Amit Vasudevan et.

all [Yer06]. El malware[1] actual es susceptible de tener métodos para modificar el código de programa que usan. Los compiladores en tiempo real (JIT) a pesar de dar un soporte transparente, no son capaces de seguir sistemas o aplicaciones multihilo, programas automodificables o código que se autotestea en modo de núcleo. Para poder resolver este problema los autores han creado una herramienta que analiza la granularidad sobre el proceso, la red, teclas usadas, ficheros, registro de sistema y un largo etcétera. Mediante esta herramienta después de detectar un comportamiento anómalo se puede analizar a través de *debuggers* las diferentes formas de un virus polimórfico, su cifrado-descifrado de datos, el contenido de memoria que ha usado.

Las técnicas de instrumentación binaria representan a la habilidad de controlar las construcciones pertenecientes a cualquier código y es empleada tanto en técnicas de análisis grueso como granular. El control se refiere a generar la construcción del ejecutable para observar una posible alteración semántica. La instrumentación binaria se clasifica en dos categorías:

- Basadas en pruebas: Dtrace [CL04], Dyninst [BH00], Detours [GH99]. Realizan su función modificando el programa objetivo para ser ejecutado de una forma determinada cuando se llama al programa original.
- JIT: Pin [LH05], DynamoRIO [Bru04], Valgrind [NS03]. Estos métodos se basan en la instrumentación mediante empleo de máquinas virtuales (VM).

Los métodos basados en pruebas no son transparentes debido a que las instrucciones originales se sobreescriben en memoria por medio de lo que se denominan trampolines[2].

El sistema que denominan SPiKE ofrece una API sobre la que integrar las funciones para los analizadores que se quieran. Introduce en la memoria virtual una serie de elementos lógicos que se activan cuando se produce una lectura, escritura o ejecución. El producto se basa en otro entorno denominado VAMPiRE [Yer06] para poder insertar esos códigos lógicos en la posición adecuada de memoria. La técnica de activación de eventos que usan, denominada por ellos *Drift Invisible* lo que hace es insertar código en unas determinadas posiciones correspondientes a la construcción del código del que se desea realizar la instrumentación. Un *BreakPoint* en código

[1] Término genérico que hace referencia a virus, troyanos, software espía (spyware)

[2] Un trampolín es una secuencia de código que se puede atravesar entrando en múltiples partes de la misma dado que no realiza operaciones que varíen el contexto del procesador.

permite detener la ejecución de código arbitrario durante el tiempo de ejecución. Para evitar que se detecte este tipo de ganchos se usan páginas de memoria que contienen la información referente a ellas mismas como "no presente" y se basan en el sistema de Detección de Excepciones de Paginado en Memoria (PFE).

Los errores más comunes de programación de aplicaciones con los que se suele ganar el control de un sistema o de una aplicación residen en la manipulación de corrupciones en memoria de los procesos. Los ataques más comunes actualmente son debidos a *Buffer Overflow* o a *Format String*. Se considera importante identificar automáticamente cuándo se han producido estos errores. El punto que presentan los autores de este texto [XNK+05] en el que se puede detectar el error producido es el momento del cierre inesperado y de forma incorrecta de la ampliación. Una vez conocidos los valores de datos y direcciones que se han embebido en el ataque se puede bloquear este tipo de ejecuciones. En el documento [XNK+05] se presenta y desarrolla un sistema descentralizado y auto protegido para los ordenadores cableados en red. Se pueden encontrar numerosos intentos para la detección de firmas de ataque de este tipo, como pueden ser los siguientes:

- Honeycomb [KC03a]
- Autograph [KK04]
- Early Bird [SEVS04]
- Poligraph [NKS05]

Pero este tipo de métodos suelen de una limitación, que generalmente se hace uso de un análisis sintáctico sobre los ataques, olvidando la semántica en la que se han producido los mismos. Muchas otras aproximaciones que se están desarrollando se basan más en la instrumentación del código binario e incluso pueden recuperarse de los ataques una vez producidos:

- TaintCheck [NS05], realiza un análisis dinámico mediante una emulación binaria del programa para atrapar la propagación del binario por la red. A su vez también se realizan firmas de ataques basadas en semántica.
- DIRA [SC05], hace una instrumentación del código para mantener un log de actualización de la memoria mientras el programa está en ejecución, realizando incluso una vuelta a atrás para volver a una zona segura cuando se ha producido el ataque.

- STEM [SSK05] realiza una aproximación reactiva. Después de detectar un ataque, se reemplaza la vulnerabilidad del programa con una versión parcheada.

Las herramientas anteriores aun siendo un paso importante introducen debido a la instrumentación de código una sobrecarga importante en la ejecución de un determinado proceso.

El análisis del flujo de la información dinámica (DIFA) [MP05] es muy útil para la detección de intrusiones a nivel de aplicación. Uno de los efectos más habituales de los ataques sofisticados es el hecho de poder introducir información insegura dentro de partes importantes de la aplicación, esta modificación es un hecho contrastable del suceso de este tipo de ataques. Muchos de estos patrones de comportamiento, una vez detectados se pueden introducir en un sistema de detección de intrusiones basado en firmas.

2.3.4 **Detección basada en firmas**

Arbaugh en el año 2001 [BAMF01] indica que en el mundo de la seguridad informática de los sistemas de alta disponibilidad, los administradores se encuentran en la tesitura de mantener sistemas a sabiendas con vulnerabilidades. Esta situación se produce debido a que se descubren fallos de seguridad continuamente, pero pasa un tiempo hasta que se tiene la solución de los mismos, conllevando que estos administradores se encuentren indefensos durante periodos de tiempo dado que una baja en el servicio conlleva pérdidas difícilmente cuantificables.

Para ello Arbaugh diseñó, implementó y evaluó un sistema que soporte la construcción y ejecución de predicados que representen una determinada vulnerabilidad. El sistema que presentan, Intro-Virt, se vale de técnicas basadas en introspección de máquinas virtuales para monitorizar la ejecución de la parte software, aplicaciones y sistema operativo. Si bien, el hecho de que una vulnerabilidad sea observada, no implica que inmediatamente se corrija, aquí entran en juego diferentes motivaciones de los administradores, desarrolladores y directivos [HJWZ04, AFM00, BAMF01, BAC+02].

La idea sobre la que se basan los autores es comprobar mediante una serie de predicados si se dan las propiedades para una u otra vulnerabilidad. Así mismo, combinando los predicados con los resúmenes de las máquinas virtuales creadas, el usuario podrá conocer si ha sucedido alguna anomalía de este tipo en el pasado. Los autores dejan claro que descubrir vulnerabilidades acaecidas, no es prevenir de una posible intrusión, pero sí es un

punto de partida para conocer hasta qué punto el sistema ha sido comprometido.

Otro de los pilares fundamentales que tienen en mente los autores es que mediante la combinación de predicados de vulnerabilidad específicos, junto con una respuesta estratégica se puede alertar al usuario e incluso prevenir sobre los ataques que están sucediendo en el sistema.

Los predicados que buscan los autores son fáciles de escribir una vez que es conocida la vulnerabilidad e incluso podrían ser transcritos por la entidad que se encargue de escribir el parche. Un predicado correcto no puede tener ni falsos positivos ni falsos negativos, pero en contraposición con los detectores de *exploits* específicos se lograría capturar las variantes del código malicioso, cualidad que no es soportada por estas últimas herramientas.

Los predicados con las definiciones de vulnerabilidades deberán ser ejecutados fuera del entorno del sistema, es decir, que no es apropiado mantener la revisión de los mismos ni en zona de usuario, ni en zona de kernel. La solución pasa por tener hardware dedicado a este tipo de tesituras.

Una de las primeras pruebas que realizan es ejecutar el software objetivo en una máquina virtual, de esta forma se obtiene acceso total al sistema en estudio, esta técnica se denomina: introspección de máquinas virtuales [GR03]. Para poder demostrar estas ideas los autores se basan en VMM (Virtual Machine Monitor), que en plataformas de virtualización como Xen [PBW03], es la parte encargada de gestionar los recursos utilizados. Así mismo si el atacante consigue poner en el peligro el propio núcleo del sistema de virtualización, debido a una fuga de la caja estanca en la que se estaba trabajando, se tendrán problemas complejos de detectar y solventar.

2.3.5 **Detección de ataques contra la lógica de las aplicaciones**

Como ya hemos visto en secciones anteriores, la detección de anomalías basadas en el análisis de trazas de llamadas al sistema de los diferentes programas para obtener un modelo de su comportamiento y posteriormente analizar sus llamadas en busca de anomalías, puede dar resultados óptimos en la detección de ataques. No obstante, este tipo de detección no es apropiada cuando lo que se ataca en la lógica de las aplicaciones. En estos casos se hace insuficiente en análisis de flujo de llamadas al sistema, pues estos ataques en contadas ocasiones modifican la trayectoria de ejecución de los procesos (con lo que la traza de llamadas al sistema invocada

por éstos se mantiene intacta). Por tanto, las aproximaciones basadas en el análisis de secuencias de llamadas al sistema no detectarían estos ataques [FHSL96b, WD01].

Otro problema es que la información que se necesita para detectar ataques a nivel de lógica de aplicación puede no estar o ser muy difícil de extraer de las llamadas al sistema invocadas por el proceso.

Para detectar ataques de éste tipo, es recomendable realizar una auditoría selectiva en ciertos puntos del flujo de control de la aplicación. El problema es que pocas aplicaciones proveen de un *hook*[1], e incluso cuando éstos están disponibles no suelen estar colocados en el lugar correcto. Además, la técnica de instrumentación suele ser específica para cada aplicación y difícil de portar a otros programas.

2.3.5.1 **Auditando los datos**

Auditar es el mecanismo de recogida de datos respecto a la actividad de los usuarios y las aplicaciones. Ya que el sistema operativo es considerado como una entidad confiable debido a que se encarga de los accesos a los recursos (como memoria y ficheros), la mayoría de los sistemas de auditoría están implementados dentro de él. Estos sistemas de auditoría no están diseñados específicamente para la detección de intrusiones y por lo tanto, en muchas ocasiones los datos producidos por los sistemas de auditoría del sistema operativo contienen información irrelevante y escasez de datos útiles. Debido a este hecho, usualmente los sistemas de detección de intrusiones desarrollan herramientas propias para acceder directamente al sistema operativo en busca de la información realmente relevante.

Lunt [Lun93] sostiene que es necesario disponer de aplicaciones auditoras que provean de la información puramente necesaria para la detección de intrusiones.

2.3.5.2 **Reescritura binaria dinámica**

Esta técnica es independiente de plataforma y soporta la recogida de información en cualquier lugar dentro del flujo de control de la aplicación y es utilizada en conjunto con firmas específicas de aplicación para detectar ataques que explotan los errores de la lógica de la aplicación. Esta aproximación ha sido validada a través de muchos estudios de casos de ataques contra aplicaciones reales como por ejemplo Apache y OpenSSH [ZV04].

[1] Sistema de recolección de datos a bajo nivel, como por ejemplo las llamadas al sistema invocadas y los argumentos de éstas.

La reescritura binaria dinámica es una técnica de post-compilación que cambia directamente el código de los ejecutables. Al contrario que la librería de interposición dinámica, la reescritura binaria dinámica funciona con programas vinculados estática y dinámicamente. Además provee de un mayor acceso a la parte interna de la aplicación ya que además de las librerías de funciones, pueden ser utilizadas funciones estáticas de la aplicación. Existen dos tipos de técnica de reescritura binaria:

1. Estática: Modifica la imagen residente en disco del binario.

2. Dinámica: Cambia la imagen de memoria de un proceso.

ATOM [SE94], EEL [LS95], Purify [HJ92], y Etch [TRB97] son ejemplos de herramientas basadas en técnicas de reescritura estática. Dyninst [BH00] y Detours [GH99] son ejemplos de herramientas de reescritura dinámica.

Comparando ambos métodos, la escritura dinámica tiene la ventaja de que mantiene intactos los ejecutables de las aplicaciones. Además, utilizando la escritura dinámica es posible modificar los binarios de la aplicación de varias maneras, dependiendo del contexto de invocación de la aplicación y permite mantener el código tras una llamada fork()[1], por lo que el proceso hijo también será instrumentalizado.

En términos generales, la reescritura binaria dinámica es una aproximación de interposición. La técnica de interposición a las llamadas al sistema ha sido utilizada para desarrollar perfiles de usuario y depuración de sistemas software, además de ser muy utilizada con propósitos de seguridad:

1. Curry, usa interposición dinámica de librerías para profiling[2] y traceo de llamadas a las librerías [Cur94].

2. Deteurs, una herramienta de depuración y profiling que utiliza reescritura binaria dinámica para interceptar funciones Win32 [GH99].

3. ByPass, extiende la funcionalidad del software de sistemas existente para computación distribuida utilizando técnicas de interposición en librerías compartidas [TL01].

[1] Se trata de una llamada al sistema operativo para que duplique el proceso que ha llamado, creando por tanto un proceso hijo que será una copia de él mismo.

[2] Técnica de creación de perfiles de usuario, aplicaciones, ...

4. Interposition Agents, es un RootKit[1] para interponerse entre el código de usuario y el núcleo del sistema operativo [Jon93].

Por lo tanto ésta aproximación separa las rutinas de auditoría de la aplicación original. De esta manera puede ser añadido un nuevo módulo de auditoría sin que se requiera la recompilación de la aplicación. La selección del módulo de auditoría es pospuesta hasta el momento en el que la aplicación es invocada, permitiendo una selección de configuración de auditoría flexible. Además, las rutinas de auditoría y la aplicación corren bajo el mismo espacio de direcciones con rendimiento comparable al acercamiento de modificación del código.

2.3.6 **Detección basada en Secuencias de Syscalls**

Estudio de secuencias de llamadas al sistema para clasificar las intrusiones [MB00] y los fallos inducidos en procesos privilegiados de UNIX. La clasificación es una capacidad esencial para responder a una anomalía, dado que permite asociar respuestas apropiadas para responder a cada anomalía. Los autores [JBDCM01] afirman que la creación de un Diccionario de Anomalías es imprescindible en este tipo de sistemas. Este tipo de diccionarios se encargan de contener la información sobre las secuencias anómalas para cada tipo de anomalía. A su vez, en paralelo con este tipo de diccionarios se pueden crear los denominados Diccionarios Normales, para poder realizar la clasificación debidamente.

Los esfuerzos durante más de 20 años se han centrado en la generación de Sistemas de Seguridad de la Información aplicando técnicas de Minería de Datos, Aprendizaje de Máquinas, Reconocimiento de Patrones e Inteligencia Artificial [JBDCM00, WL99, RKMJ00, NP99]. La metodología planteada en [JBDCM01], denominada *stide*[2] se puede resumir de la siguiente forma:

1. Modelado: Se caracteriza el fenómeno en estudio por medio de una secuencia de variables o símbolos dentro un alfabeto largo pero finito. Estas secuencias se obtienen manteniendo el programa en estudio ejecutándose desde el arranque hasta el cierre.

[1] Es una herramienta que se ejecuta a nivel del sistema operativo capaz de interceptar llamadas al sistema y afectar a los procesos que se están ejecutando, con diversos propósitos.

[2] *Sequence time-delay embedding.*

2. Recolección de Datos Normales: Recogida de grandes volúmenes de datos correspondientes a la operación normal del programa, almacenando los símbolos generados. Ene este apartado se realiza el estudio numerosas ocasiones, produciendo numerosos procesos, hilos... tratando de explorar la operación normal de funcionamiento de todas las formas posibles.

3. Extracción de características: Se selecciona un tamaño de cadena (k), y se construye un diccionario de secuencias cortas sobre los datos del Diccionario Normal. Esto se realiza mediante una ventana deslizante del tamaño k a través de la muestra, saltando sólo un único símbolo cada momento.

4. Detección: Dado una muestra de entrada del sistema en estudio, se realiza la detección de anomalías mediante comparación de secuencias contra el fichero salvado de normalidad anterior. Si las secuencias no son encontradas se denominarán "secuencias anómalas" y se almacenarán en el Diccionario de Anomalías.

2.3.7 Detección basada en grafo

2.3.7.1 Detección basada en grafo de llamadas al sistema

Los autores establecen que si bien el kernel de linux dota al usuario de una serie de funciones interesantes de modularidad y flexibilidad, en sistemas embebidos se puede lograr un mejor rendimiento, para ello se valen de lo que han denominado como "grafo de llamadas al sistema" basado en la ingeniería inversa de este tipo de sistemas. Mediante una representación en forma de grafo se puede representar el comportamiento del sistema objetivo mejorando o haciendo hincapié en optimizar determinadas funciones.

Un grafo de llamadas es un grafo dirigido que representa la estructura estática del programa. Al igual que el grafo anterior se ha realizado para un programa, se puede realizar un grafo semejante para las llamadas desde el código fuente de las librerías. Teniendo el grafo adecuado para cada máquina, dependiendo de las aplicaciones que se tengan instaladas, se puede diferenciar un comportamiento normal del mismo o sucesos anormales. En la misma línea, dado que los sistemas de detección de intrusiones se usan para detectar los rastros de actividades maliciosas sobre redes y sistemas, en el documento de Mutz de 2006, [DMry] se indica una forma para detectar una secuencia de llamadas al sistema anómalas.

Los sistemas de detección de intrusiones se clasifican, como ya se ha comentado en puntos anteriores, de dos formas: basados en malos usos o basados en anomalías. En los primeros sistemas (misuse-based) [Pax98, U.99] se tiene un número de descripciones de ataque, o firmas, que se tratan de localizar sobre flujos de datos de auditoría de sistemas. Estos sistemas generalmente son eficientes y tienen una baja tasa de detecciones erróneas que se denomina "falsos positivos". Si bien, la gran pega de este tipo de sistemas es el hecho de que no detectan ataques que no haya sucedido alguna vez y de los que se tenga conocimiento. Un buen ejemplo para una tarea concreta puede ser el caso de detección de condiciones de carrera, en el que se ha desarrollado un lenguaje de auditoría de trazas BSM, denominado BSML [PUR05]. En este documento se explica un lenguaje para determinar cuando se dan estas situaciones y actuar en consecuencia por si se esta produciendo un ataque.

Dentro de los diferentes tipos de sistemas de detección de intrusiones, los que se basan en anomalías [Den87, GC98] construyen modelos del comportamiento que se espera de las aplicaciones. Para generar estos modelos se revisa la operación normal de las mismas, guardándolo como punto de referencia. Los modelos que se consiguen se denominan "perfiles".

De la forma anterior comparando los modelos de comportamiento de cada aplicación con el comportamiento actual se pueden obtener las desviaciones que hayan surgido. Una de las premisas de esta forma de actuación es el hecho de que una desviación sobre el comportamiento aprendido, conlleva ser víctima de un ataque. En el documento de Denning [Den87] se trata de realizar una aproximación basándose en perfiles de usuario, pero este tipo de técnicas no pueden ajustarse a los cambios de repentinos de los usuarios en su comportamiento.

Se han desarrollado diferentes sistemas de anomalías con diferentes técnicas de detección [JV91]. La secuencia de llamadas al sistema producida por las aplicaciones ha sido objeto de análisis de detección de anomalías. Las técnicas que se han ido proponiendo son aproximaciones de dos tipos: basadas en especificaciones y basadas en aprendizaje.

La aproximación mediante especificaciones confía en unos modelos específicos para cada aplicación que han sido realizados manualmente [KRL97, BM02, CC02], bien con conocimiento experto, bien derivadas a través del uso de algún programa con alguna técnica de análisis [WD01]. Los perfiles generados de esta forma son introducidos como entrada de un sistema de detección de intrusiones en tiempo real, así cuando se produzca una secuencia de llamadas no conformes con el modelo se levantará la corres-

pondiente alerta. La mayor pega de este tipo de sistemas de detección es el hecho de que tienen una limitada capacidad para generar las especificaciones, tarea normalmente bastante ardua para ser realizada a mano. Otro problema de este tipo de HIDS es la necesidad de interacción del usuario durante la fase de entrenamiento, si bien es posible utilizar modelos predefinidos para cada aplicación, no son aplicables para todos los usuarios, sobretodo al emplear diferentes configuraciones. Así mismo, para este tipo de aproximación se requiere el acceso al código fuente.

Si ahora se analiza la técnica de aprendizaje se debe señalar que no es necesario asumir nada de antemano, si no que los perfiles se construyen en base al análisis de las llamadas al sistema durante la ejecución normal. Esta primera aproximación vino de la mano de Forrest [FHSL96b]. La clave de este método es el hecho de que un programa que tiene que interaccionar con el sistema operativo en el que esta siendo ejecutado requiere hacer uso de las llamadas al sistema para poder causar un daño permanente en el sistema. Cuando una secuencia de llamadas al sistema se desvía del comportamiento esperado, se asume que se está produciendo un ataque.

La idea era recabar durante la fase de aprendizaje todas las posibles secuencias de llamadas al sistema de una determinada longitud. Durante la fase de detección, se comparaban las cadenas de secuencias recogidas con las legítimas aprendidas en la primera fase, teniendo la posibilidad frente a desviaciones de indicar la anomalía. El problema de esta técnica es que sólo hace uso de la secuencia que siguen estas llamadas, dejando a un lado los argumentos y los valores devueltos al retornar de las mismas. Además en la metodología de la autora sólo se examina un proceso en concreto.

Siguiendo esta línea de trabajo, otros autores han continuado el trabajo precursor del análisis de secuencias de syscalls [WD01, WP99, FKFL03], y por norma general es la opción más utilizada para análisis del comportamiento de los programas.

Las técnicas basadas en detección de anomalías conlleva un riesgo inherente de detectar como anomalía comportamiento de secuencias que son legítimas en el funcionamiento de una determinada aplicación, así como realizar código malintencionado que siga las secuencias de llamadas correctas. Además generalmente las técnicas que se aplican no hacen uso de los argumentos que llevan asociados las llamadas al sistema.

Se proponen dos posibles mejoras a los sistemas de detección de anomalías basados en host:

1. Aplicar sistemas de detección que incorporen los argumentos que lle-

van las llamadas al sistema.

2. Descubrir una forma de cuantificar la puntuación de anomalía de cada modelo en una medida global. Este resultado determinará si un evento es parte o no de un ataque.

En concreto para resolver este problema los autores proponen una técnica que hace uso de las Redes Bayesianas para realizar la clasificación de las llamadas al sistema. Los autores demuestran que el análisis de los argumentos y el uso de clasificadores Bayesianos mejoran la precisión contra los intentos de evasión. La salida de cada uno de los motores de detección individuales se combina con una Clasificación Bayesiana, mejorando a los clasificadores basados en umbrales naive que se usan para combinar las salidas de los modelos.

El sistema propuesto al estar basado en análisis de llamadas al sistema individuales, es más resistente contra los ataques que tratan de asemejarse a la forma normal de comportamiento de una aplicación: ataques "*mimicry attacks*" [TM02a, TM02b, WS02]. La definición formal de este tipo de ataques es aquel que mediante código inyectado ataca a la seguridad de un sistema emulando la secuencia de llamadas al sistema de las aplicaciones legítimas.

El modelo que presentan los autores [DMry], consiste en un flujo ordenado de llamadas al sistema almacenadas dentro del sistema operativo $S = \{s_1, s_2, ...\}$. Cada llamada al sistema $s \in S$ tiene un valor de retorno r^s y una lista de argumentos $< a_1^s, ..., a_n^s >$. Se debe indicar que los autores no están teniendo en cuenta las relaciones entre las llamadas al sistema. Para cada llamada al sistema creada por una aplicación se crea un perfil, comparando después del aprendizaje cada llamada de esa aplicación a las *syscalls* del sistema si concuerdan con el perfil aprendido en la primera fase.

En el modo de detección, la función de cada modelo es devolver la probabilidad de que ocurra un determinado argumento basándose en el modelo aprendido previamente. Un valor de baja semejanza ("*likelihood*") de que una determinada característica es observable en un perfil establecido indica la posible anomalía que se esta produciendo.

Para los autores del documento [DMry] sólo se considera ataque aquello que se salga de lo común dentro de los posibles parámetros que se puedan tener en un argumento de llamada al sistema. Si un ataque se realiza sin invocar a las llamadas del sistema con unos argumentos dentro de los del modelo, no será reconocido. Para poder caracterizar las llamadas al sistema

se aplican una serie de modelos que ayudan a la identificación de anomalías:

1. *Longitud de la cadena de texto*. Generalmente los parámetros que se introducen en las llamadas a las API's o al sistema ejecutivo tienen una longitud máxima de unos cientos de caracteres y generalmente son legibles directamente por el ser humano. Si se introduce un ataque dentro de este tipo de parámetros se acaban insertando caracteres no reconocidos que aumentan los tamaños normales que se tenían dentro de cada parámetro.

2. *Distribución de Caracteres*. La forma en la que se distribuyen los caracteres dentro de las cadenas de texto de los argumentos suele ser regular, aparte de ser generalmente caracteres legibles. Por normal la mayor parte de los caracteres se encuentran en un pequeño conjunto de 256 caracteres: letras, números y unos poquitos caracteres especiales. Dentro de los lenguajes naturales escritos, por ejemplo castellano o ingles, los caracteres tienen una distribución de frecuencias de aparición uniforme. Es cierto que no se puede comparar un lenguaje natural con el empleado por un sistema computerizado, pero tienen ciertas semejanzas en la distribución de caracteres que bien pueden ser tenidas en cuenta.

3. *Inferencia estructural*. Es posible que el atacante haya conseguido mantener la acción dentro de unos parámetros normales para la distribución de caracteres, y/o en una longitud adecuada a la "normalidad" de las longitudes de cadenas de texto. Para poder capturar adecuadamente este tipo de ataques se realiza el análisis de la estructura de argumentos, que no es más que la gramática de expresiones regulares que puede emplear ese argumento en concreto. La inferencia estructural es el proceso por el cual se obtiene esta gramática.

4. *Buscador de Tokens.* El objetivo del localizador de tokens es determinar si los valores de determinados argumentos de llamadas al sistema se pueden resumir en un conjunto de alternativas posibles reducido. Esto se debe a que generalmente una aplicación pasa a menudo valores idénticos, como flags y handlers, en los argumentos de las syscalls. Cuando un ataque cambia el flujo normal de ejecución y consigue saltar al código malicioso propiamente dicho, estas restricciones son violadas.

Finalmente para homogeneización de los resultados de los diferentes motores se emplea una clasificación basada en redes bayesianas que tiene la estructura de la figura 2.5.

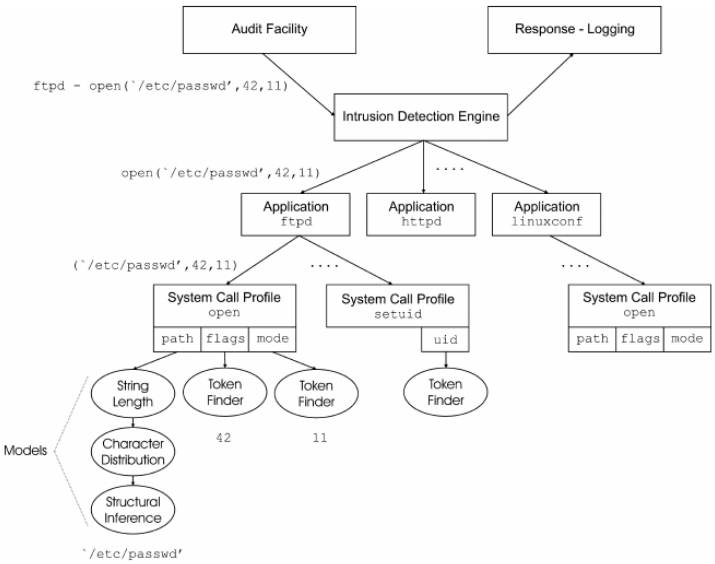


Figura 2.5: Arquitectura bayesiana del sistema de detección de anomalías para host.

Siguiendo la línea de investigación abierta para superación de los sistemas de detección más utilizados basados en detección de desviaciones, hay autores que clasifican estos sistemas como "caja blanca", "caja negra" o "caja gris" [GRS04a]. Para discernir los sistemas y poder clasificarlos se valen de la información que usan para construir el modelo mediante el que realizan la comparación:

- Los detectores de caja negra emplean únicamente el número de la llamada al sistema utilizada. Potencialmente estos detectores pueden hacer uso de los argumentos en esas llamadas.
- Los detectores de caja gris extraen además información en tiempo de ejecución a medida que las llamadas al sistema se van produciendo.
- Los detectores de caja blanca extraen el modelo directamente desde el código fuente o del binario.

Los detectores de caja negra y caja gris ya han sido comentados en el documento. Los detectores de caja blanca se basan en el concepto de falsos

positivos cero, dado que siempre indican una posible intrusión. Si bien estos detectores no son siempre apropiados, en principio debido a que el código fuente no es siempre accesible, además que el hecho de realizar un análisis estático es complejo por causas externas: ofuscación, sistemas de protección digital, packers[1]... Otro inconveniente de las cajas blancas son los programas que se cambian a sí mismos, muy comúnmente usados en técnicas víricas.

En el paper [GRS04a] exponen una nueva técnica que denominan grafo de ejecución, que es técnica pionera juntando las tres cajas. Se trata de construir un modelo que acepte las mismas secuencias de llamadas al sistema pero con las bases de partida de las cajas blancas. Los autores denominan a la secuencia de observaciones como "ejecución", basándose en llamadas a API's intermedias o a syscalls directamente de sistemas UNIX, en los que se va apilando/rellenando los parámetros en la pila para hacer la llamada o en los registros para solicitar la interrupción directamente. Así mismo en la misma traza de ejecución se anexan los valores de retorno y las direcciones de retorno que se requieran. Finalmente el concepto de grafo de ejecución, que se construyen en base a las observaciones de ejecución explicadas con anterioridad, consta de pares de identificación de llamadas al sistema consecutivas y las direcciones de retorno de la pila cuando se realiza la llamada en sí misma. Dado que cada una de las direcciones devueltas revela información sobre la estructura de llamadas del sistema, se puede considerar una aproximación a la terminología de caja blanca.

```
   int main(int argc, char *argv[]) {
1:    int a, b;
2:    a = 1; b = 2;
3:    f(a);
4:    g();
5:    f(b);
   }

   void f(int x) {
1:    sys_call(5);
2:    if (x == 1)
3:       sys_call(3);
4:    else if (x == 2)
5:       sys_call(4);
   }

   void g() {
1:    sys_call(2);
   }
```

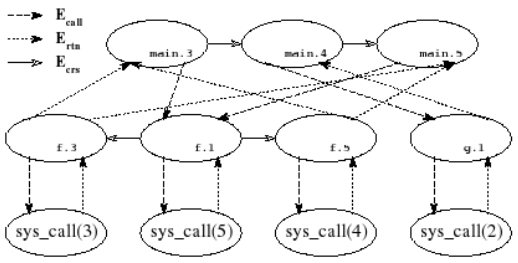


Figura 2.6: Código fuente y grafo de ejecución.

[1] Un packer es un programa, generalmente comercial que se encarga de impedir que se pueda realizar ingeniería inversa en los programas binarios que se distribuyen.

2.3.8 Visualización e identificación del contexto de las intrusiones desde las trazas de llamadas al sistema

Los sistemas de detección basados en anomalías son muy útiles pues son capaces de detectar técnicas de intrusiones nuevas que no han sido analizadas y que no tienen firmas conocidas. Sin embargo esta capacidad conlleva el problema de la posibilidad de generar falsos positivos. Esto es debido a que el análisis de la normalidad se lleva a cabo mediante el análisis del funcionamiento del programa y comportamientos normales pueden quedar fuera de éste análisis. La mayoría de las técnicas tratan de generalizar el comportamiento normal para modelar mejor este tipo de conducta. Aun así este hecho implica que se contemplará mayor parte de una posible pauta intrusiva. La identificación del contexto de la intrusión[1] puede ayudar significativamente a aumentar la tasa de detecciones y a reducir los falsos positivos generados [ZL04].

2.3.8.1 Flujo de trabajo del Identificador del contexto de intrusiones (ICI)

La adhesión de un módulo de identificación del contexto de intrusiones consiste en colaborar con el motor de análisis de anomalías, realizando un aprendizaje off-line, (es probable que al módulo ICI le lleve un tiempo considerable determinar los contextos de intrusiones), en el cual cada vez que salte una alarma el módulo ICI analizará el contexto y determinará si ésta era un falso positivo o no. En caso de serlo la secuencia que ha hecho saltar la alarma debería ser añadida al sistema inteligente de detección de intrusiones como parte del comportamiento normal, evitando de esta manera futuros falsos positivos por el mismo motivo.

Estas premisas detallan el proceso en el que estaría involucrado el módulo ICI [ZL04]:

1. Todos los eventos procesados por el motor de análisis son copiados al módulo ICI para que los guarde en un buffer de tamaño predefinido.

2. Toda alarma generada por el motor de análisis también es enviada al módulo ICI.

3. El módulo ICI extrae el contexto de la intrusión de la alarma, y más adelante lo procesará, visualizará y/o lo agrupará.

[1] En la detección de intrusiones basada en anomalías, el contexto de la intrusión que corresponde a una alarma, son las huellas auditadas que se encuentran fuera del modelo normal del comportamiento las que hacen saltar la alarma

4. El módulo ICI busca contextos de intrusión ya identificados similares.

5. Un experto en seguridad analiza el contexto de la intrusión y decide si la alarma ha sido un falso positivo o no y el nivel de confianza teniendo en cuenta los contextos existentes.

6. El vector ICI (alarma,decisión,confianza) de la alarma es enviado al motor de análisis y/o a los sistemas de respuesta.

7. El motor de análisis afina el modelo de comportamiento normal utilizando el vector obtenido del ICI para evitar volver a dar el mismo falso positivo.

8. Los sistemas de respuesta activan diferentes estrategias en base al vector ICI.

2.3.8.2 **Otras utilidades para el ICI**

Además de la capacidad para reducir los falsos positivos, el módulo ICI puede ser utilizado como una herramienta para analizar tareas como las que se enumeran a continuación:

1. Para fortalecer la correlación de alertas con las técnicas de detección de anomalías ya que una alerta no puede determinar consistentemente una intrusión y el contexto de la intrusión es capaz de mostrar las razones exactas.

2. Para estudiar las características de las intrusiones y mejorar después en base a la información obtenida la eficiencia del experto y proponer nuevas técnicas de análisis.

3. Para aplicarlo en otros campos de la intrusión como el estudio forense.

2.3.8.3 **Visualizando y analizando el contexto**

Sabemos que las anomalías en una secuencia de llamadas intrusivas estarán indicadas por un set de secuencias extranjeras. Por tanto sería de gran ayuda visualizar dicha secuencia no propia del programa en el esquema del ICI. El tamaño de la ventana aplicado para dicha aplicación también será crítico ya que en el caso de que la secuencia extraña sea mayor que el tamaño de la ventana, ésta no será totalmente detectada. En los gráficos producidos para la visualización del contexto de la intrusión la longitud

de todas las secuencias intrusivas deben ser mostradas de modo que se conozca si existe la posibilidad de detectarlas completamente o no en base al tamaño de la ventana. Este tipo de grafo llamado Gráfico de secuencias extranjeras o anómalas [ZL04] (FSG - *Foreing secuence graph*).

Finalmente se deberían tratar de identificar secuencias anómalas mínimas (MFS) y los eventos con los que están asociadas para que sean identificados como parte del contexto. Para lograrlo, los datos pueden ser analizados mediante dos métodos:

1. Separando los datos en bloques más pequeños y evaluando cada bloque.

2. Evaluando todo evento que se encuentre en los datos.

El primer método puede llegar a romper las secuencias anómalas, por lo que la utilización del segundo método es más acertada.

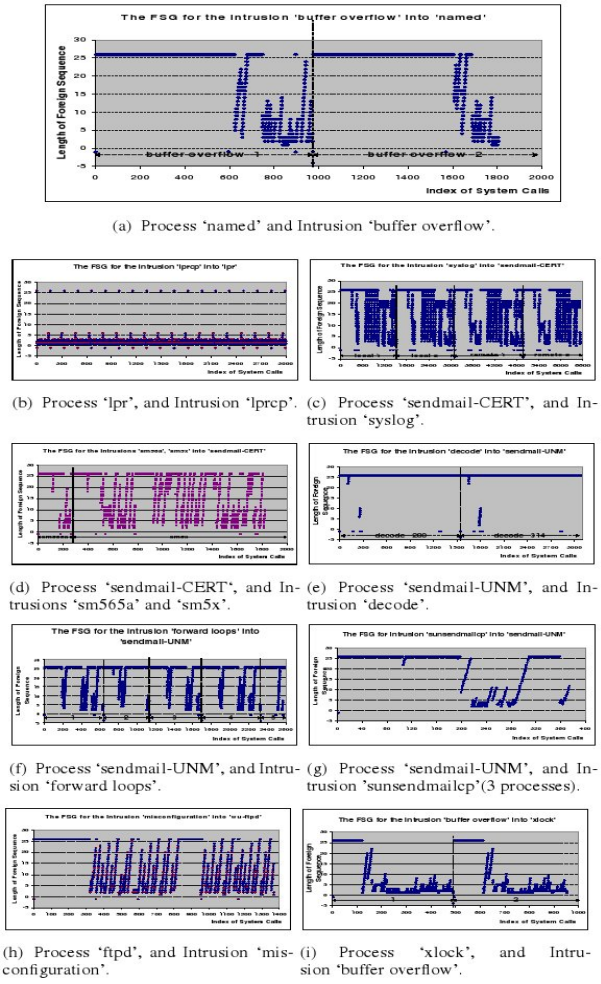
La siguientes imágenes muestran un ejemplo [ZL04] de visualización de un FSG. Los datos provienen de la Universidad de Nuevo México y también han sido utilizados en [WP99], figura 2.7

Normal Datasets	Intrusive Datasets	No. of Traces	No. of System Calls
live-named-UNM	—-	142	9230572
—-	buffer overflow-1	3	969
—-	buffer overflow-2	2	831
live-lpr-MIT	—-	2703	2926304
—-	lprcp	1001	165248
sendmail-CERT	—-	294	1576086
—-	syslog-local-1	6	1516
—-	syslog-local-2	6	1574
—-	syslog-remote-1	7	1861
—-	syslog-remote-2	4	1553
—-	cert-sm565a	3	275
—-	cert-sm5x	8	1537
sendmail-UNM	—-	346	1799764
—-	decode	36	3067
—-	forward loops	36	2569
—-	sunsendmailcp	3	1119
syn-wu-ftpd	—-	8	180315
—-	misconfiguration	5	1363
syn-xlock-UNM	—-	71	339177
—-	buffer overflow-1	1	489
—-	buffer overflow-2	1	460

Figura 2.7: Set de datos utilizados para el FSG.

Para analizar las características de cada intrusión, su set de datos, incluso siendo en el mismo proceso es tratado como set de datos individual

puesto que cada intrusión genera sus propias características de contexto en los rastros revisados.



(a) Process 'named' and Intrusion 'buffer overflow'.

(b) Process 'lpr', and Intrusion 'lprcp'.

(c) Process 'sendmail-CERT', and Intrusion 'syslog'.

(d) Process 'sendmail-CERT', and Intrusions 'sm565a' and 'sm5x'.

(e) Process 'sendmail-UNM', and Intrusion 'decode'.

(f) Process 'sendmail-UNM', and Intrusion 'forward loops'.

(g) Process 'sendmail-UNM', and Intrusion 'sunsendmailcp'(3 processes).

(h) Process 'ftpd', and Intrusion 'misconfiguration'.

(i) Process 'xlock', and Intrusion 'buffer overflow'.

Figura 2.8: Ejemplo de FSG para diferentes procesos.

Para éste gráfico FSG, el número de secuencias anómalas en cada set de datos es representativo y la mayoría de las intrusiones pueden ser detectadas aplicando un tamaño de ventana adecuado. Otro fenómeno significativo es que las intrusiones no generan secuencias anómalas en la primera fase (excepto en la subfigura b). Por ejemplo en la imagen (a), tras 600 llamadas al sistema se han generado secuencias de llamadas anómalas y esto de alguna forma es indicador de que existe un periodo de "calentamiento" para las intrusiones, al menos para aquellas que son detectadas mediante sistemas basados en anomalías.

Además en las tres instancias de "sunsendmail" en la subfigura (g) se genera el mismo gráfico FSG. Esto quiere decir que tienen las mismas características de intrusión. Al mismo tiempo, para las diferentes instancias de la misma intrusión en las subfiguras (a),(c),(f) o (i) sus gráficos FSG también son similares. Este fenómeno puede beneficiar investigaciones en las técnicas de detección de anomalías puesto que no es necesario crear técnicas específicas para detectar diferentes ejecuciones de una misma intrusión.

Para diferentes intrusiones en un proceso, los gráficos FSG son diferentes; para el proceso "*sendmail*" desde CERT, el gráfico FSG de la intrusión en la subfigura (c) es diferente del gráfico FSG para la intrusión sm565a o sm5x en la subfigura (d); el gráfico FSG de las intrusiones en el mismo proceso "*sendmail*" (e), (f) y "*sunsendmailcp*" en la subfigura (g) también son diferentes entre si. Este hecho prueba que no existen estrategias generales para detectar todas las intrusiones en un proceso, y que las estrategias de técnicas específicas de detección son necesarias para detectarlas.

2.3.8.4 **Identificando el contexto de las intrusiones: MFSs**

Las secuencias anómalas mínimas en los rastros auditados son una de las características producidas por una intrusión, lo cual es utilizado por detectores de anomalías para detectar la intrusión. Como se ha mencionado antes, los MFS son el contexto de las alarmas. Basado en la identificación del contexto de la intrusión en los set de datos, sus estadísticas numéricas son resumidas en tablas.

Por último la información de las tablas debería disuadir posibles ataques del tipo mimicry [WS02] y del paradigma de ocultamiento de información [TKM02], visto el gran número de secuencias anómalas mínimas reportadas, en especial, para la intrusión de "buffer overflow".

2.3.9 **Modelación de llamadas al sistema mediante ventana deslizante**

Este método extiende las primeras investigaciones [ELSS01] en la detección de anomalías en llamadas al sistema para la detección de intrusiones, incorporando tamaños de ventana dinámicos. El tamaño de la ventana es la longitud de la subsecuencia de una traza de llamadas al sistema las cuales son utilizadas como la unidad básica para el modelado del comportamiento de una aplicación o de un proceso. Se presentan dos métodos para la estimación óptima del tamaño de la ventana basados ambos, en la disposición

de datos para el entrenamiento [EGS00]. El primer método es un modelado de la entropía el cual, determina el tamaño óptimo de la ventana para los datos. El segundo método es una función modeladora de probabilidad que tiene en cuenta el contexto para establecer el tamaño de ventana. Un tamaño de ventana dependiente del contexto está motivado por el hecho de que las llamadas al sistema son generadas por los procesos.

2.3.9.1 **Modelo de entropía**

Se puede utilizar un marco teórico de información para escoger el tamaño de ventana óptimo para las llamadas al sistema. En este marco, se estima cual de los diferentes tamaños de ventana realizará mejor la ordenación de los datos mediante computación. Una descripción detallada sobre la aplicación teórica de información a la detección de anomalías es presentada en [GRS04b].

La premisa básica para la detección de intrusiones es esa ordenación intrínseca en los datos auditados que es consistente con el comportamiento normal, y distinto del comportamiento anómalo. Se mostrará empíricamente que cuantos más datos ordenados existen mejor es el rendimiento del modelo de detección de anomalías. El proceso de construcción de un modelo de detección de anomalías debería por tanto involucrar una primera medida de orden de los datos bajo los diferentes modelos. En este caso, se está interesado en medir la ordenación bajo diferentes tamaños de ventana. Esta ordenación puede ayudar a elegir el mejor tamaño de ventana.

Para modelar las llamadas al sistema, se usa un modelo predictivo a través del cual la probabilidad the "n" llamadas al sistema es estimada por las "n-1" llamadas previas. La forma en que la predicción del modelo es entrenada, es que por cada "n-1" secuencia de llamadas al sistema presentes en la traza[1], se mantienen contadores de las siguientes llamadas. La predicción para una determinada llamada al sistema dada una secuencia de "n-1" llamadas previas es, el número de esa llamada dividido por el número total[2] de llamadas. En este punto la longitud "n" es el tamaño de la ventana del algoritmo que modela. Se medirá la ordenación de los datos para la predicción del modelo bajo diferentes valores de "n". Se propone utilizar una medida teórica de los datos para medir la ordenación de los

[1] Una traza de llamadas al sistema es una secuencia de todas las llamadas al sistema que un proceso de una aplicación dada realiza durante su tiempo de vida (ejecución).

[2] Para evitar probabilidades 0, al contador de cada tipo de llamada al sistema se le añade un pequeño valor.

datos, entropía condicional, para ayudarnos a elegir el valor de "n". Cuantos más datos normalizados, menor entropía. La definición de la entropía condicional esta reglejado en la ecuación 2.12.

$$H(x|y) = -\sum_{x,y \in X,Y} P(x,y) \log_2 P(x|y) \tag{2.12}$$

donde P(x,y) es la probabilidad conjunta de "x" e "y", y P(x|y) es la probabilidad condicional de "x" dado "y". En el contexto de la llamadas al sistema se deja que $X = \sum$ sea el set de llamadas al sistema e $Y = \sum^{n-1}$ sea el set de secuencias de llamadas al sistema de longitud "n-1". "D" será la notación para el total de secuencias en la traza, $|(x,y)|$ será el número de veces que se origina la secuencia en "D", y $|D|$ será el total de números de secuencias en la traza. Se puede definir entonces la probabilidad conjunta como $P(x,y) = \frac{|(x,y)|}{|D|}$ La probabilidad condicionada de $P(x|y)$ es la predicción de este modelo. Desde que $P(x|y) = 0$ para una secuencia (x,y) que no ocurre en la traza y la entropía condicional es aditiva, se puede representar la entropía condicional para una ventana de tamaño "n" de forma más eficiente para la computación, ecuación 2.13.

$$H_n(X|Y) = -\sum_{x \in \sum, y \in \sum^{n-1}} P(x,y) \log_2 P(x|y) = -\sum_{(x,y) \in D} \frac{1}{|D|} \log_2 P(x|y) \tag{2.13}$$

2.3.9.2 **Grafo de llamadas de una aplicación**

La motivación para una dependencia del contexto del tamaño de la ventana proviene del mecanismo subyacente de cómo un proceso se ejecuta. Las llamadas al sistema en la traza dependen de la trayectoria en la ejecución del proceso. Asimismo, la trayectoria de un proceso depende de muchos factores como entradas recibidas o el estado del sistema donde se está ejecutando. Estos factores determinan que trayectoria de ejecución seguirá el proceso en cada desviación posible.

Se podrían modelar todos los set de las posibles trayectorias de la aplicación utilizando "grafos de llamadas[1]". Los grafos de llamadas modelan la estructura de la aplicación y define las posibles trayectorias de la ejecución. Cada nodo representa una posible bifurcación del programa, y las aristas

[1]Es un gráfico en el cual cada ruta define una posible trayectoria de ejecución para un proceso o una aplicación.

están etiquetados con las llamadas al sistema que se dan entre los nodos. Existe un nodo de comienzo definido para el grafo y al menos un nodo final. Es posible observar la trayectoria de ejecución de un proceso a través de el grafo de llamadas del sistema asociado a éste y por lo tanto una traza de llamadas al sistema sería simplemente las llamadas al sistema a lo largo de la trayectoria de ejecución del proceso.

Se debe tener en cuenta que a pesar de que estos grafos existan para todos los programas, obtenerlos es muy costoso debido a que variarán por el código fuente, por el compilador y por el sistema operativo empleados. Debido a ésto, es prácticamente inviable asumir que se pueden recrear los grafos a partir de las llamadas al sistema observadas. Aún no pudiendo determinar el grafo específico de llamadas, se puede asumir que éste existe y utilizar ésto para motivar un tamaño de ventana dependiente del contexto.

En la aplicación del grafo de llamadas al sistema a la detección de intrusiones, existe un set de trayectorias de ejecución que corresponden a exploits[1]. El objetivo del método de modelado de llamadas al sistema es el ser capaz de determinar si una secuencia de llamadas al sistema corresponden a una trayectoria de ejecución normal, o si bien pertenece a la trayectoria de ejecución de un exploit. Hay que tener en cuenta la longitud de la secuencia ha sido ajustada en el contexto del grafo de llamadas y que con mayor probabilidad una secuencia larga puede identificar unívocamente una sección del grafo. Sin embargo, es usualmente demasiado larga para ajustarse a una sola arista por lo que debe abarcar más ramas. Para que estas secuencias sean observadas varias veces, los estados de los diferentes procesos donde tienen lugar las secuencias más largas deberán forzar la trayectoria de ejecución para que coincida con estas ramas. Desgraciadamente ésto puede generar ruido en el modelo.

Por otro lado, las secuencias más cortas abarcan menos ramas, sin embargo, pueden tener lugar en múltiples puntos del grafo de llamadas resultando complicado determinar unívocamente dónde surge la secuencia y, por tanto, si ésta pertenece a un trazado normal o a el trazado de un exploit.

Idealmente, para una secuencia dada se prefiere escoger aquella que sea la más corta y que identifique unívocamente la rama del grafo donde se genere. Debido a que los nodos de las ramas ocurren en lugares diferentes, la longitud óptima de una secuencia depende específicamente de las llamadas al sistema que ésta contiene. Por lo tanto, la longitud óptima depende del

[1] Es el nombre con el que se identifica un programa informático malicioso, o parte del programa, que trata de forzar alguna deficiencia o vulnerabilidad de otro programa.

contexto.

2.3.9.3 **Sparse Markov Transducers (SMT)**

Los SMT[EGS00] son utilizados para modelar trazas de llamadas al sistema estimando una probabilidad "predictiva" dependiente del contexto motivada por el marco del grafo de llamadas. Esta es la probabilidad de predecir la siguiente secuencia de llamadas al sistema dada la secuencia previa. Una vez que se ha entrenado al modelo desde datos basados en la normalidad de un proceso, se dispone de una distribución de probabilidad predictiva sobre dicho proceso. En la fase de evaluación de secuencias de llamadas al sistema para determinar si entran dentro del rango de la normalidad o si bien pertenecen a una traza generada por la ejecución de un exploit, se computa cada la probabilidad predictiva para cada secuencia. Si la probabilidad obtenida está por debajo del umbral establecido entonces la probabilidad de que la secuencia haya sido originada por el proceso en funcionamiento normal es muy baja y la traza será evaluada como exploit (anómala). El establecimiento del umbral define la diferencia entre falsos positivos y el ratio de detección del sistema

2.3.9.4 **Detección basada en grafos N-gramas de llamadas al Sistema**

Este tipo de sistemas basados en cadenas de Markov con N-gramas[Mar01], se engloban en el área denominada como computación inmunológica. para ello basándose en el documento de Forrest [FHSL96b] han introducido la idea de los n-gramas para caracterizar el modelo. Para poder establecer el valor de la N se deben basar los cálculos en la granularidad de las llamadas al kernel, que puede variar de un sistema operativo a otros.

La caracterización del comportamiento de un programa al igual de lo propuesto por Forrest se basa en la recogida de segmentos deslizantes de longitud N. Los autores indican que un valor de N grande implica un coste computacional alto y así mismo, un valor bajo será inválido para la caracterización del sistema.

Otra de las partes importantes del estudio es la determinación del momento en el que se puede dar el aprendizaje por finalizado, dado que un aprendizaje incompleto puede llevar a estimar una tasa de falsos positivos altos. Para poder estimar de forma correcta, durante el principio del aprendizaje la base de datos crece rápidamente, sin embargo cuando la tasa de incremento desciende y se hace muy pequeña, se puede decir que la base

de datos ha convergido. En este momento se podrá concluir con que la base de datos es completa y se puede dar por finalizado el aprendizaje.

Para poder construir la base de datos con cadenas de texto de múltiples longitudes se puede tener en cuenta las siguientes consideraciones:

1. Se construye el árbol de sufijos para los N-gramas de los datos de aprendizaje para algún valor que sea suficientemente grande de N. N será el límite superior en la longitud de cadenas multilongitudinales. Una máquina de estados finita (FSM) mediante el algoritmo "two-finger" se puede construir de forma inmediata a partir de ese sufijo.

2. Se puede compactar el árbol de sufijos en un grafo dirigido acíclico mediante la fusión de subárboles equivalentes.

3. Incluso se pueden llegar a anexar dos subárboles si son muy parecidos introduciendo una pequeña variación en la forma de compactar.

En general un árbol de sufijos para una determinada cadena de caracteres contiene todos los posibles sufijos de la misma, por lo que un árbol de sufijos de longitud m tiene m sufijos diferentes. El árbol de sufijos para un conjunto de cadenas contiene cada sufijo de cada una de ellas. Así mismo el árbol de sufijos para un conjunto de N-gramas puede construirse realizando primero el árbol de las cadenas de entrenamiento y después entroncando cada rama con la profundidad N deseada.

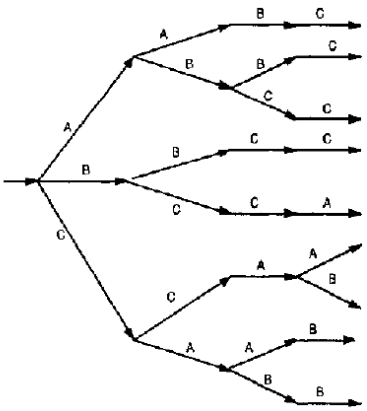


Figura 2.9: Un árbol de sufijos para el ejemplo de la cadena A B C C A B B C C A A B C

Cada nodo en profundidad $k < N$ se denomina k-grama, los nodos hijos de profundidad N se llaman N-gramas y el nodo raíz se rellena con una cadena vacía. Cada enlace se etiqueta con un símbolo. Estos enlaces de sufijos

conectan cada nodo a su sufijo apropiado más largo. Los sufijos más largos proporcionan una forma de ir de un N-grama (un nodo hijo) al siguiente N-grama:

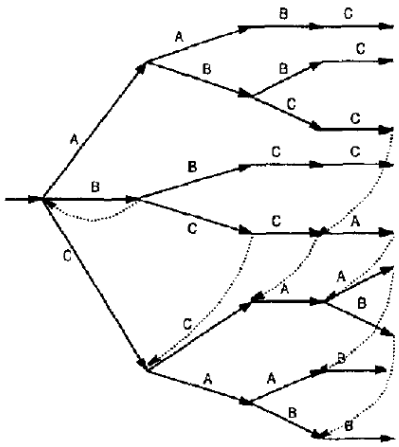


Figura 2.10: Árbol de sufijos con punteros de sufijos incluidos.

A veces puede ser necesario incluso incluir un nodo denominado "Unknown_symbol" (nodo desconocido) para poder contemplar los sucesos que se salgan fuera de lo normal en el modelo representado por el árbol.

El algoritmo definido como detección de "two-finger" es como leer la cadena "ABCCABBCCABABC" con dos dedos de la mano, dejas el dedo de la izquierda fijo al principio de la cadena mientras desplazas el derecho. El desplazar un caracter el dedo derecho es como descender un nivel en el árbol. Como se requiere que los dedos estén separados como mucho por N letras, cuando se alcanza este momento se desplaza una posición el dedo de la izquierda, luego si en la cadena para una longitud de 4 el máximo era ABCC, ahora se tendrá BCC para continuar y posteriormente BCCA. Una anomalía en la cadena de llamadas al sistema requiere un desplazamiento del dedo izquierdo.

Con esta representación del modelo de cadenas de llamadas al sistema se puede realizar el mismo desarrollo efectuado por Forrest [FHSL96b]para el análisis de las trazas del sistema y detección de anomalías sobre las mismas.

2.3.9.5 **Detección mediante el uso de Autómatas en Línea**

Una nueva forma de abstracción en el comportamiento de un programa es la denominada "Inlined Automaton Model (IAM)"[RG05a]. Una vez se

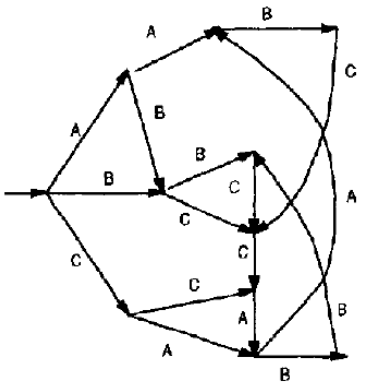


Figura 2.11: Árbol final resultante de la compresión de la figura anterior.

define el modelo de comportamiento de un determinado programa (accediendo al código fuente) se obtiene un grafo como el ejemplo de la figura 2.12.

```
main( int argc, char** argv) {
   int fd;
   if ( argc == 1 ) {
     write(1, "StdOut", 6);
     foo(1);
   } else {
     fd = open(argv[1], O_WRONLY);
     foo(fd); close(fd);
}  }
void foo(int x) {
   write(x,"Hello World",11);
}
```

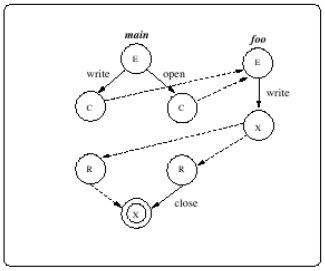


Figura 2.12: Ejemplo de código y representación de representación NFA.

Si en base a lo anterior se materializa la forma en la que se ejecuta, como es el caso que se esta planteando, se obtiene la figura 2.13.

La monitorización de los programas se basa en las llamadas a las librerías de funciones, con lo que se tienen que desarrollar un sistema de interposición que se encargue de comparar las llamadas con el modelo que se tiene. Los mayores problemas que se pueden plantear en este tipo de situaciones son derivados de la recursividad. El hecho de que el código fuente de un programa para diferentes casuísticas vuelva sobre sí mismo, deja un campo abierto para que se tomen por buenas numerosos cambios en el mismo que darían un resultado de comportamiento normal, teniendo un ataque en ejecución, figura 2.14 y 2.15

La granularidad es importante y un problema a considerar en este tipo de sistemas de detección. Idealmente un IDS debería monitorizar todas

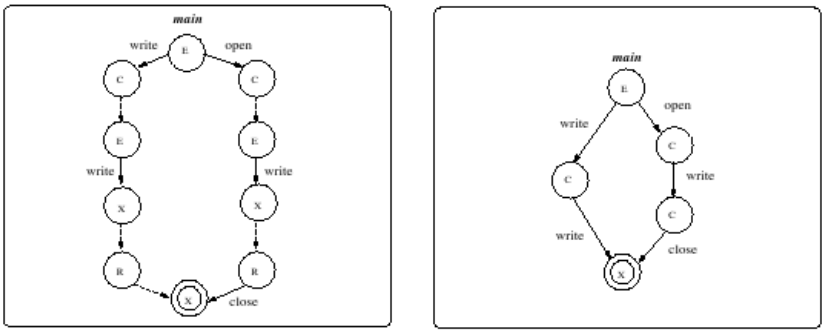


Figura 2.13: Ejemplo de implementación de IAM.

```
main( int argc, char** argv) {
   if (argc > 1) foo(--argc, argv);
}
void foo(int argc, char **file) {
   int fd;
   if ( argc != 0 ) {
     fd = open(file[argc], O_WRONLY);
     write(fd,"Hello World",11);
     foo(--argc, file); close(fd);
  }
}
```

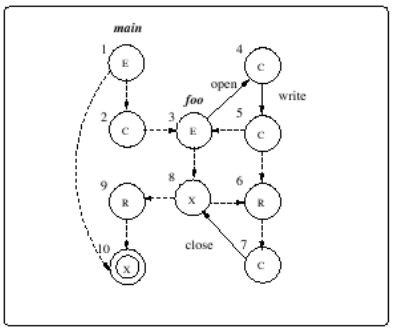


Figura 2.14: Ejemplo de recursividad y autómata generado.

y cada una de las sentencias ejecutadas por el programa en estudio y de esa forma, validar cada instrucción de la máquina. Esto por lo general no es posible, lo que lleva a afrontar el problema desde diferentes puntos de granularidad: cuanto más gruesa la aproximación más sencillo será generar un ataque mimicry. Por ejemplo, restringir exclusivamente los eventos observables a llamadas al sistema, significa que las llamadas a las librerías no son incluidas en el modelo. Numerosos ataques de "buffer overflow" y "format string" han acaecido contra las propias librerías instaladas en el sistema operativo.

2.3.9.6 **Detección mediante el uso de Autómatas en llamadas a la pila**

Hay otro trabajo [LB05], que se basa a su vez en autómatas, que incluye en los mismos la información de la pilla de llamadas. Este modelo se publicita como más potente sobre los que se basan exclusivamente en secuencias de syscalls, detectando ataques adicionales que los anteriores no detectan. Este

```
main( int argc, char** argv) {
   if (argc > 1) fool(argc, argv);
}
void fool(int argc, char **file) {
   if ( argc != 0) foo2(--argc, file);
}

void foo2(int argc, char **file) {
   int fd; fd = open(file[argc], O_WRONLY);
   write(fd,"Hello World",11);
   fool(argc, file); close(fd);
}
```

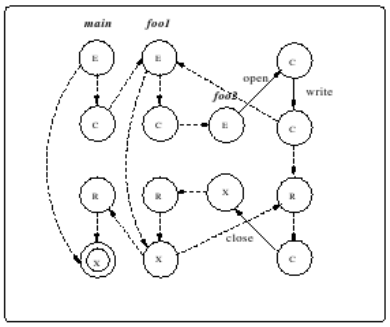


Figura 2.15: Ejemplo de recursividad indirecta y autómata generado.

método se basa en el descrito por Feng et al. [FKFL03], pero realizando su función de forma más general. Aun basándose en autómatas de apilamiento de llamadas (HPDA), en vez de hacer uso de una pila extra como en el modelo de PDA de Wagner and Dean [WD01], este motor de análisis se basa exclusivamente en la pila del sistema operativo correspondiente. Cuando se invoca a una syscall, el sistema de detección de intrusiones se activa y comprueba si se puede pasar del estado en que se encuentra al siguiente. El modelo HPDA, parecido al PDA es una 5-tupla: (S, Σ, T, s, A) donde S es un conjunto finito de estados y $Sigma$ es un conjunto finito de símbolos definido como $\Sigma = (YxAddr) \cup \epsilon$ siendo $Y = \{Entry, Exit, Syscall\}$y Addr es el conjunto de posibles direcciones definidas por el programa ejecutable. Las entradas de este tipo de modelos son de los siguientes tipos:

- Entrada. Es un punto de entrada para las llamadas del sistema. La dirección asociada es la dirección de retorno de esta función.
- Salida. Es un punto de salida de una función de llamada del sistema. La dirección asociada es también la dirección de retorno de esta función.
- "Syscall" es la invocación de una llamada al sistema, donde la dirección asociada es la dirección de retorno de esta llamada.
- ϵ es el string vacío.
- T es el conjunto de funciones de transición $(Sx\Sigma \rightarrow S)$
- s es el estado inicial $(s \in S)$
- A es el conjunto de estados aceptados $(A \subseteq S)$

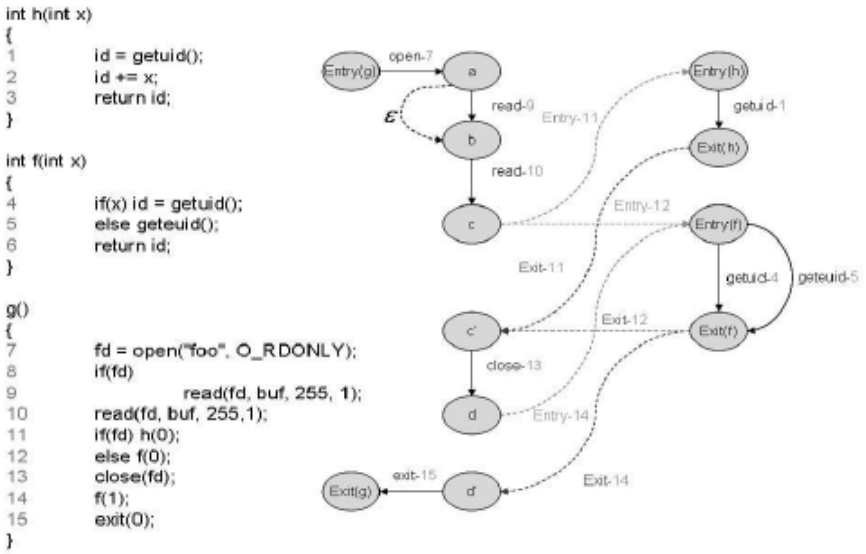


Figura 2.16: Ejemplo de código y modelo HPDA.

2.3.10 **Modelado de trayectorias de syscalls mediante análisis del código fuente**

Uno de los inconvenientes a la hora de trabajar con sistemas de detección basados en anomalías del comportamiento "normal" de las aplicaciones, es que no se dispone de todas las muestras posibles. Ésto tiene como consecuencia que un número, (normalmente muy bajo si el entrenamiento ha sido adecuado), de trayectorias de ejecución no es representado en el modelo creado y por lo tanto, la posibilidad de generación de falsos positivos[1] existe desde un principio. La razón por la que un entrenamiento puede no se correcto, radica en la dificultad técnica de obtener todas las trayectorias posibles de una aplicación ya que su comportamiento se captura durante su ejecución. Además en los sistemas basados en anomalías se fijan umbrales de normalidad debido a que pueden existir comportamientos normales que pueden ser tan inusuales como un comportamiento intrusivo. Mediante esta medida se evita pasar comportamientos intrusivos por alto, pero en cambio se desligitimiza por segunda vez funcionamientos poco probables

[1] Un falso positivo se da en el momento que el sistema de detección de anomalías detecta una trayectoria de ejecución, (traza de llamadas al sistema), y genera una alerta siendo dicha trayectoria no intrusiva, (se encuentra dentro del comportamiento normal de la aplicación).

aunque correctos de la aplicación.

Wargner y Dean [WD01] introdujeron por primera vez la idea de extraer del código fuente de las aplicaciones un modelo de las trayectorias de ejecución. En ejecución cualquier llamada al sistema que se desvía del modelo determinado estáticamente es considerada como una intrusión y por lo tanto debe ser prohibida. Un gráfico de llamadas derivado de un gráfico de control de flujo (CFG) es un autómata finito no determinístico (NFA) debido como construye el control en forma de $if-then-else$[1] y funciones $call/return$[2]. El grado de indeterminismo en un CFG determina el número de trayectorias imposibles [WD01] y de esta manera latitud disponible para los ataques de tipo mimicry [WS02], los cuales son capaces de evadir los sistemas que inspeccionan patrones de llamadas al sistema generando los patrones de llamadas que la aplicación secuestrada debería generar hasta que localizan el tipo de llamada al sistema que necesitan. Wagner [WD01] y Giffin [JTGM02] trataron de utilizar un modelo autómata de apilación inversa (PDA push-down automaton) para tratar de mitigar el problema de trayectorias imposibles. Aunque el modelo de PDA es más preciso que el de NFA, incurre en una costosa comprobación en tiempo de ejecución [WD01, JTGM02]. Sin embargo el desafío en la prevención de intrusiones basada en llamadas al sistema es reducir la vulnerabilidad a ataques mimicry al tiempo que se minimiza la sobrecarga en tiempo de ejecución.

2.3.10.1 **Construcción del modelo de llamadas al sistema**

Recientes sistemas de detección[WD01, FKFL03, RSD01, SAHS98, JTGM02, WP99, LC04] definen modelos de comportamientos normal utilizando la actividad realizada en tiempo de ejecución. Los primeros trabajos como [RSD01, SAHS98] utilizan perfiles para construir modelos de comportamiento normal. Sin embargo, generar los perfiles requiere tiempo y puede producir modelos incompletos. La manera más idónea de construir el modelo es aplicar un análisis estático para extraer patrones de llamadas al sistema de la aplicación.

La manera más sencilla de extraer un modelo de llamadas es partiendo del grafo de llamadas propuesto por Wagner y Dean [WD01], quienes directamente deducen el grafo de llamadas del grafo de control de flujo (CFG)

[1] Sentencias condicionales que en base al resultado de la evaluación de una condición booleana (positivo o negativo) ejecutan unas acciones u otras.

[2] Instrucciones de llamada y retorno a una función. En el retorno se da el control a la siguiente instrucción desde donde fue hecha la llamada.

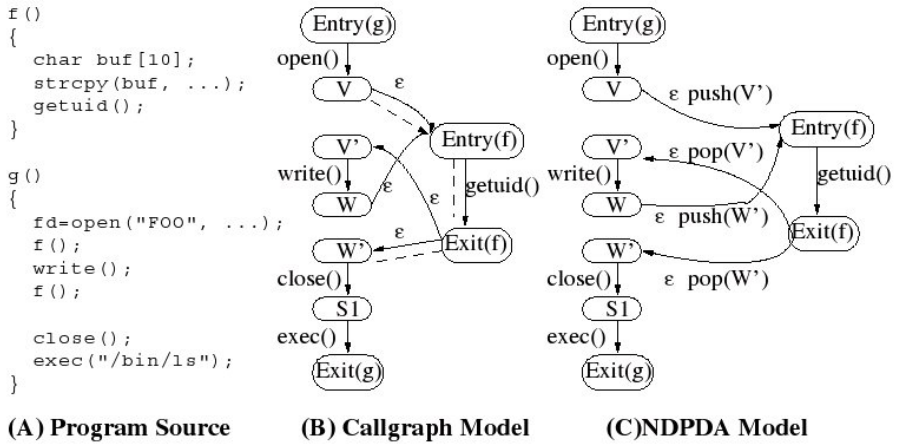


Figura 2.17: Ejemplo de un programa en C asociado a un modelo de grafo de llamadas (NFA) y un modelo NDPDA.

abstrayendo todo a excepción de las funciones y los nodos de llamadas al sistema como se muestra en la subfigura (B). El CFG resultante es un autómata de estados no determinista de NFA debido a que las construcciones tales como $if - then - else$, los bucles y las funciones son llamadas desde múltiples sitios.

Una limitación en el modelo de grafo de llamadas es el problema de rutas imposibles como es indicado por la linea intermitente en la subfigura (B), la cual surge porque la función f es llamada desde dos lugares, V y W. El grafo permite una trayectoria imposible $\{v- > Entrada(f)- > Salida(f)- > W'\}$, lo cual no es posible. La consideración de este tipo de rutas como posibles tiene como consecuencia que exista una posibilidad para un ataque de tipo mimicry [WS02]. Por ejemplo, si asumimos que ocurre un ataque por desbordamiento del buffer en la función f en la figura (A), cuando f retorne la ejecución sería transferida al código intrusivo inyectado mediante el desbordamiento. Al tomar el control, el código intrusivo generaría la secuencia de llamadas $\{close(), exec(''/bin/sh'')\}$, logrando llevar a cabo el ataque pues la secuencia generada se corresponde con una de las posibles en el grafo (B). Esta limitación de los modelos de grafos de llamadas es debido al hecho que sólo verifica el orden entre llamadas al sistema definidas como la aplicación, pero no su emplazamiento.

Para eliminar las trayectorias imposibles, Wagner simula las operaciones de pila utilizando un autómata no determinista de apilación inversa (NDPDA) [WD01] como se representa en la subfigura (C). La idea clave del modelo NDPDA reside en que cuando se encuentra una llamada desde V,

el estado de retorno correspondiente V' es introducido en una pila simulada antes de que el control sea transferido, y es retirado cuando la llamada devuelve el control. Este funcionamiento asegura que la función siempre retornará al lugar donde ha sido llamada sin importar los lugares desde donde ha sido llamada. Sin embargo, para cada llamada al sistema entrante, el modelo NDPDA debe calcular un set de posibles configuraciones de la pila, basándose en las llamadas al sistema previas. Este set de configuraciones puede crecer indefinidamente, especialmente en las llamadas recursivas. A pesar de un complicado algoritmo para reducir la complejidad del modelo NDPDA, el peor de los casos en tiempo de ejecución sigue siendo cúbico en la longitud de las llamadas al sistema. Esto conlleva una alta e inaceptable sobrecarga en tiempo de ejecución para aquellas aplicaciones que son de tamaño grande.

Los modelos PAID [LC04] y Dyck [JTGM04], utilizan inserciones de llamadas al sistema nulas para transformar aplicaciones y eliminar el indeterminismo. Sin embargo el modelo Dyck puede producir sobrecargas impredecibles debido a una inserción desmesurada de llamadas al sistema nulas. Más aun, el modelo Dyck es indeterminista debido a que no inserta llamadas al sistema nulas en funciones recursivas. El modelo PAID[LC04], utiliza grafos inline para eliminar el problema de trayectorias imposibles. Aunque PAID es un modelo determinista el modelo que genera es dos o tres veces mayor y requiere extensas modificaciones, figura 2.18.

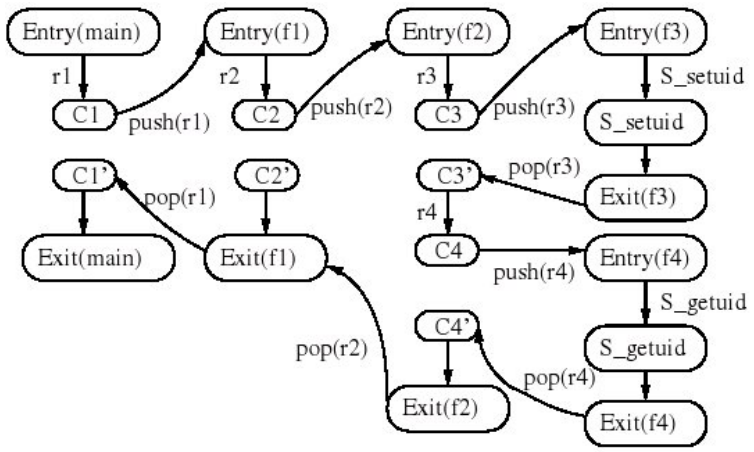


Figura 2.18: Proceso de llamada a *getuid* utilizando el modelo VSL.

Cuando *getuid* es llamado, el VSL previo es $\{r_1, r_2, r_3\}$, el VSL actual es $\{r_1, r_2, r_4\}$, y el prefijo de ambos VSL es $\{r_1, r_2\}$. Por tanto, tras consumir r_3, el algoritmo pasa el estado actual de S_setuid a $C3'$, y después a $C4$ tras

consumir r_4. Finalmente, introduce r_4 en la pila y mueve el estado actual a $Entry(f4)$ [LLC06].

Feng propuso un modelo estático (VPS) [BGM02], el cual explota el estado de la pila de usuarios para identificar cada lugar de llamada al sistema. Para cada llamada al sistema entrante extrae todas las direcciones que se encuentren actualmente en la pila para formar una lista de pila virtual (VSL) y alimentar con ella a un autómata determinista de apilación inversa o DPDA como se muestra en la figura anterior. Los símbolos de r_1 a r_4 en el DPDA representan la dirección de la llamada. Asumiendo que el VSL para una llamada al sistema previa Sp es $\{a_1, a_2, ..., a_1, b1, b2, ..., b_m\}$, y el VSL para las llamadas al sistema actuales Sc es $\{a_1, a_2, ..., a_1, c_1, c_2, ..., c_n\}$. La secuencia $\{a_1, a_2, ..., a_1\}$es un prefijo común para los dos VSL. Sus algoritmos transversales utilizan $\{b_m, b_{m-1}, ..., b_1\}$ para generar los símbolos de entrada que simulan operaciones de retorno de funciones, y $\{c_1, c_2, ..., c_n\}$ para generar los símbolos de entrada que simulan operaciones de llamada a funciones. Tras la llamada a *getuid* en la figura superior el VSL para *getuid* es $\{r_1, r_2\}$. Asumiendo que el estado actual es S_setuid, el algoritmo utiliza $\{r_3\}$ y $\{r_4\}$ para generar las siguientes operaciones transversales:

1. No consumir nada y mover al siguiente estado $Exit(f3)$.
2. Sacar un elemento de la pila. Si el elemento en la parte alta de la pila es r_3, mover el estado a $C3'$.
3. Consumir r_4 y mover el estado a $C4$.
4. Empujar r_4 en la pila y mover hasta el estado $Entry(f4)$.
5. Mover hasta el estado *getuid* cuyo contador coincide el contador de programa de la llamada actual *getuid*.

Si todas estas operaciones son aceptadas por el modelo DPDA entonces la llamada al sistema *getuid* actual es legítima.

Si *getuid* no es llamada, el modelo VPStatic no puede aceptar la llamada al sistema *setuid* debido a que el modelo no puede alcanzar el estado *S_setuid* desde el estado $Entry(main)$. La función $f1$en la subfigura de la derecha (B) es llamada dentro de un bucle. El algoritmo VPStatic actual no puede aceptar la secuencia de llamadas $\{setuid, getuid, setuid\}$ desde que el algoritmo sólo retorna al prefijo del actual VSL previo.

El modelo VPStatic no puede manejar el indeterminismo debido a las sentencias $if - then - else$. Por ejemplo en la subfigura superior (A), la lla-

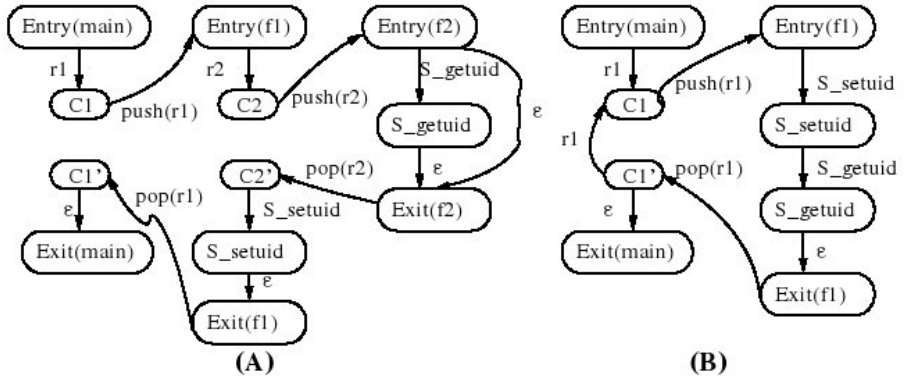


Figura 2.19: Proceso de llamada *getuid* utilizando el modelo VPStatic.

mada al sistema *getuid* en $f2$ no es siempre ejecutada debido a que se realiza dentro de una sentencia $if - then - else$. Si *getuid* no es ejecutado, el modelo VPStatic genera un falso positivo cuando *setuid* es ejecutado. En el momento que *setuid* es llamado el VSL previo es {} y el VSL actual es $\{r_1\}$. Según el algoritmo, las únicas operaciones que pueden generar para atravesar el autómata son:

1. Consumir r_1, mover al estado $C1$

2. Introducir $r_1$en la pila, mover hasta el estado $Entry(f1)$.

3. Mover hasta el estado *setuid* cuyo contador coincide el contador de programa de la llamada actual *setuid*.

Obviamente, tras la segunda operación, la tercera no puede ser aceptada por el autómata desde que $C2$ es el único estado que se puede alcanzar desde el estado $Entry(f1)$.

VPStatic también falla al manejar el indeterminismo cuando se efectúan llamadas a funciones desde bucles, como se muestra en la subfigura (B) superior. Además aunque utiliza el estado de la pila del usuario, no puede prevenir todos los ataques mimicry [LLC06]. Por ejemplo un ataque de mimicry podría llenar la pila de tal manera que llegaría a engañarle demandando una llamada al sistema mediante una instrucción trampa, y recuperando el control y repitiendo el mismo proceso. Esto es debido a que el modelo no tiene en cuenta la dirección de retorno de la instrucción trampa y el atacante puede obtener el control tras la llamada a la instrucción trampa.

El modelo PAID utiliza una combinación de todas las direcciones de retorno en la pila del usuario y la dirección de retorno de la llamada trampa que se encuentra en la pila del núcleo como coordenada unívoca para cada instancia de llamada al sistema en la aplicación. Además utiliza un algoritmo nuevo de grafo transversal que soporta backtracking[1] para tratar con indeterminismos creados por sentencias condicionales del tipo $if-then-else$ y es tan eficiente como el algoritmo transversal DFA [LLC06]. Finalmente para reducir más aún la vulnerabilidad debido al tamaño de ventana por parte de ataque mimicry, PAID realiza un extenso y restringido análisis de los argumentos de las llamadas al sistema, el cual, puede crear restricciones totales o parciales en los argumentos de llamadas al sistema basado en ficheros de configuración. Ésta habilidad, estrecha los posibles valores de los argumentos en muchas de las llamadas al sistema susceptibles de ser atacadas, tales como la *open*().

2.3.11 **Solución de instrumentación mediante criptografía**

El avance de las llamadas al sistema autenticadas [MR06] se basa en el hecho de introducir dentro de misma una serie de parámetros extra: argumentos para control de políticas, argumentos criptográficos que garanticen la integridad de la política y de los argumentos. Para poder realizar este tipo de funciones y proteger los sistemas:

1. Transformar los programas para cambiar las llamadas al sistema con llamadas al sistema autentificadas.

2. Revisión en tiempo real por parte del kernel para asegurar que cada llamada al sistema esta dentro de la política que le corresponde.

Para empezar, instalación en el sistema, se analiza el binario de un programa por un instalador de confianza, quien primero generará la política que comprenda el comportamiento de cada llamada al sistema usando un análisis estático y reescribe el binario de forma que cada syscall contenga tanto la política como la validación criptográfica que necesite.

El segundo paso, comprobación de la syscall en tiempo real, se realiza por medio de una intercepción en el kernel, que verifica el contenido criptográfico utilizado, con lo que si todo es correcto se permite la ejecución de la llamada y si no se obliga al cierre del programa solicitante.

[1] Es un método de resolución automática de problemas mediante una búsqueda sistemática de posibles soluciones. Las soluciones no válidas se eliminan y no se vuelven a comprobar.

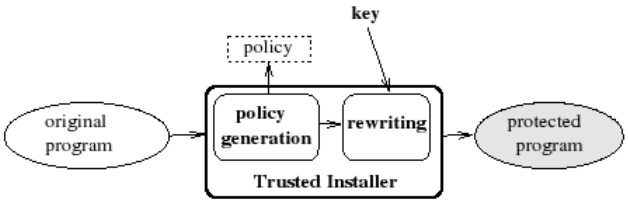


Figura 2.20: Instalación del programa.

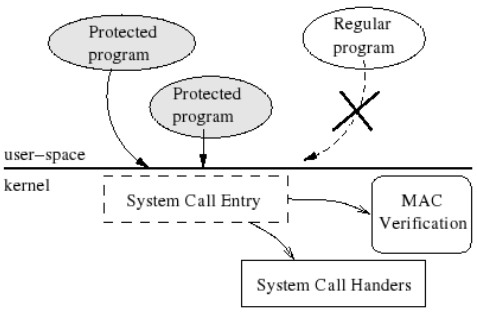


Figura 2.21: Comprobación de las syscalls.

2.4 Sistemas de Intrusión basados en fuentes heterogéneas: Big Data.

La detección de intrusiones ha sido ampliamente estudiada tanto a nivel comercial como a nivel de investigación, si bien todavía los expertos en *Cyber* Seguridad coinciden en la falta de precisión en las alertas generadas y la carencia de buenas herramientas para lidiar con las amenazas, de forma que sea más sencilla la gestión de la Seguridad en las empresas de hoy en día. Las mejoras en el área pasan por conseguir sistemas de gestión de amenazas más sencillos de usar, que mezclan las diversas fuentes heterogéneas relacionadas con las herramientas de Seguridad y permitan correlacionar los eventos de las mismas de forma inteligente.

Uno de los grandes retos que se tienen que abordar es el hecho de que cada una de las fuentes de los sistemas integrales de seguridad ya supone en sí mismo un origen de datos Big Data (por ejemplo: el tráfico de red). El problema entonces ya no es sólo el abordar el Big Data de cada uno de los sistemas de detección de intrusiones por sí mismo, si no el hecho de generar un sistema que procese Big Data heterogéneo, generado por múltiples herramientas: Network Intrusion Detection Systems (NIDS), Host Intrusion Detection Systems y Security Information and Event Management (SIEM), entre otros.

Conforme los cyber-ataques han ido evolucionando y mejorando en sofisticación, los Sistemas de Detección de Intrusiones han seguido la misma línea y se han ido haciendo cada vez más complejos, teniendo que monitorizar múltiples fuentes heterogéneas. Dentro de los productos tradicionales, actualmente, se despliegan:

1. Sistemas tipo NIDS (*Network Intrusion Detection System*), que monitorizan el tráfico de red en múltiples emplazamientos de la red corporativa.

2. Sistemas tipo HIDS (*Host Intrusion Detection System*), que monitoriza todos los Hosts de la red: logs, procesos corriendo, ficheros, accesos a la red, actividad del usuario, etc.

3. Sistemas tipo PIDS (*Protocol based Intrusion Detection System*), que monitorizan protocolos específicos, como por ejemplo el protocolo HTTP.

4. Sistemas tipo APIDS (*Aplicacion Protocol based Intrusion Detection Sys-*

tem), como por ejemplo los monitorizadores del protocolo SQL (*Structured Query Language*).

5. Firewalls, para bloquear, monitorizar y controlar el tráfico de red, tanto con el exterior, como en las diferentes sub-redes corporativas.

6. Software Antivirus desplegado a nivel corporativo. Que se ocupa de monitorizar los sistemas, como pueda hacer un HIDS, si bien está especializado en la detección de ficheros ejecutables maliciosos.

7. Sistemas de Base de Datos, con sistemas de auditoría y monitorización.

El personal IT (*Information Technology*) que se ocupa de los sistemas de seguridad, generalmente, se ve desbordado por una sobrecarga de información, muchas veces ambigua o indicando alarmas falsas. Es más, generalmente las grandes empresas emplean varios productos de cada una de las categorías anteriores, para garantizar la recogida de la información adecuada y no estar limitado a un único proveedor de tecnología, implicando que muchas alarmas ocurren varias veces, sin tener herramientas que permitan identificar esta situación.

Un hecho que se debe tener en cuenta es que incluso una única fuente de eventos de seguridad, como pueda ser el tráfico de red, ya es en sí mismo un problema de Big Data y se enfrenta a los retos de este área de trabajo. Tal y como indica Nassar et al. [NaBM13], un enlace GigaBit de tráfico de red que requiera una monitorización y un estudio de los paquetes de datos en profundidad del tráfico (DPI[1]), visto desde el punto de vista de la Seguridad, ya supone un caso de trabajo para Big Data. Otro reto Big Data que se encuentra en las organizaciones, es el hecho de que disponen de ficheros de logs enormes de cada uno de los Hosts que monitorizan. Según indica *Cloud Security Alliance* en 2013 una empresa como HP puede generar 1 trillón de eventos por día, con una media de 12 millones de eventos por segundo [ALL13]. , lo que supone un problema ya sólo para almacenar la información convenientemente. Empresas que tienen problemas como los anteriores no pueden usar las soluciones actuales, y al tener múltiples sistemas funcionando, las falsas alarmas son un problema grave, ya que no hay forma de correlacionar todas las fuentes que suelen estar en diferentes formatos. Las tecnologías basadas en bases de datos relacionales, en general, suelen ser un cuello de botella a la hora de estudiar la información. Por

[1] *Deep Packet Inspection*

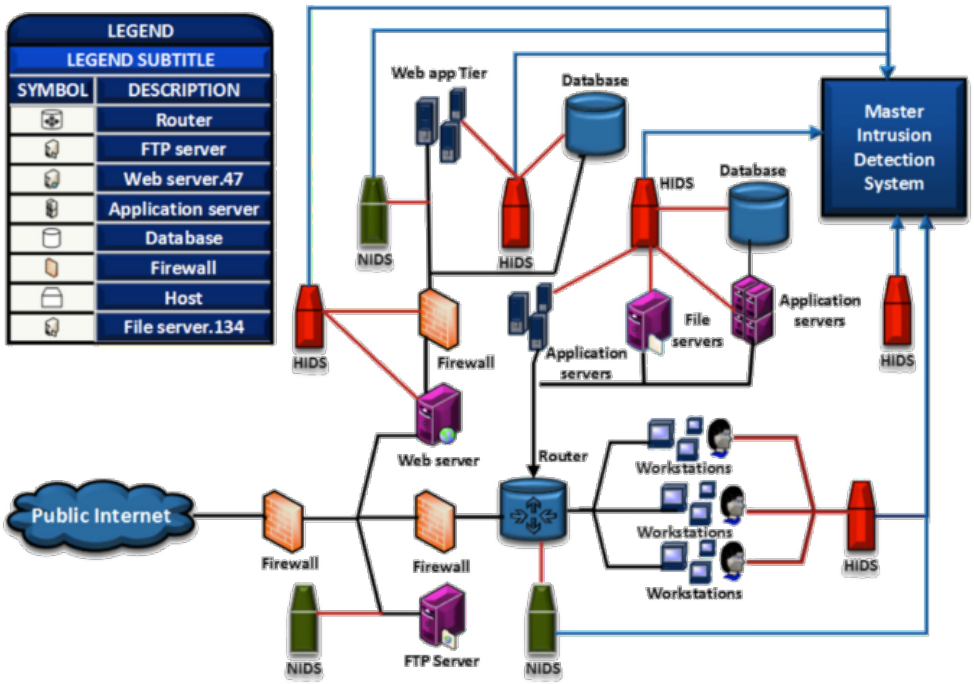


Figura 2.22: Ilustración de ejemplo de fuentes heterogéneas moitorizadas.

ejemplo, los sistemas SIEM comerciales que usan bases de datos como repositorios de almacenamiento, tal y como indica Zions Bancorporation [E.12] a la hora de buscar en los datos de seguridad del último mes se debería esperar 20 minutos, mientras que con herramientas como Hadoop se tardaría 1 minuto en conseguir los mismos resultados. Una empresa típica puede llegar a tener varios productos de seguridad que no se integran nada bien entre sí, lo que causa un problema de heterogeneidad de cara a abordar la Detección de Intrusiones. Tal y como indica Lawrence Pingree, director del Centro de Investigación Garner, este concepto se denomina *Intelligence Awareness*, que no es más que la capacidad de la inteligencia artificial de compartir y alertar a los expertos aunando múltiples tecnologías heterogéneas [E.13]. Ed Billis, CEO de Risk I/O comenta la misma situación, en la que los productos de seguridad están aislados entre sí: "Los sistemas SIEMs no se diseñaron originalmente más que para consumir información de logs (*syslog*) y recopilar información de red con unas pocas excepciones sobre la gestión de la configuración y de las vulnerabilidades. El análisis de la Seguridad es más que Big Data - son datos diversos. Este hecho causa graves limitaciones técnicas que no son fáciles de superar sólo mediante sistemas SIEM" [K.14].

2.4.1 **Antecedentes de los Sistemas de Detección de Intrusiones y el Big Data.**

El término de Big Data se define generalmente en términos de las 3 Vs: Volumen, Velocidad y Variedad; tal y como planteó Doug Laney en 2001 [Lan01]. El volumen se refiere a la cantidad de datos y sólo se aplica el término de Big Data en aquellos casos en los que las grandes cantidades de datos representan un problema a la hora de afrontar su tratamiento con las tecnologías tradicionales. La velocidad implica a la tasa de recepción y procesamiento de la nueva información, siendo Big Data aquellos casos en los que el flujo de datos varía muy rápidamente para tratarlo con las tecnologías tradicionales. La variedad en cambio se refiere a la complejidad de los datos, y se habla de Big Data cuando los datos incluyen problemas complejos: alta dimensionalidad, datos de varias fuentes, datos con diferentes estructuras y/o formatos. Pese a que hay más definiciones del término de Big Data, como las 5 Vs de Zikopoulous [ZPD+12] que añade la Veracidad y el Valor, por simplicidad, se tratará el término de Big Data como la computación en cualquier momento de flujos de datos que no pueden ser abordados mediante las tecnologías tradicionales.

Desde el punto de vista de la detección de intrusiones, ya en el año 1994, Frank [Fra94], centrado en la reducción de datos y la clasificación de los mismos se encontró con que un usuario genera entre 3 y 35 MB de datos en un periodo de 8 horas, conllevando varias horas de computación para analizar los datos más relevantes para una única hora. Las conclusiones del estudio fueron que era imprescindible el filtrado, la clusterización y la selección de características para mejorar la precisión de las detecciones.

Tal y como se quería indicar, se puede entender la información de los Sistemas de Detección de Intrusiones como Big Data, ya que cumple con el Volumen de información proveniente de múltiples fuentes de forma que no se pueda tratar con la tecnología tradicional, tiene una gran tasa de eventos por segundo que se deben recoger y procesar para dar respuestas en tiempos razonables, y dispone de gran variedad de flujos de información provenientes de múltiples fuentes con distintos formatos.

Desde un punto de vista de monitorización, las mejoras en el sector de la Seguridad no sólo implican el uso de Sistemas de Detección de Intrusiones. Estos sistemas generalmente son complementados con Sistemas de Prevención de Intrusiones (IPS) [Gro15], que como es lógico requieren de una detección en casi tiempo-real. Julish y Dacier [JD02] indican que el 99% de las alarmas que se generan en un IDS son falsos positivos; Xu y

Ning [XN08] argumentan que en los términos de tasas de detección, los IDSs son poco precisos, teniendo un número de falsos negativos inaceptable, perdiendo ataques reales. La detección de intrusiones es un problema de Big Data de acuerdo con Suthaharan y Panchagnula [SP12]. Bhatti et al. [BLB+12] presentan el hecho de que actualmente ni con las tecnologías más actuales se cubren las necesidades de Detección de Intrusiones para Big Data: "*El análisis de la Seguridad dentro de un entorno Big Data presenta unos retos únicos, no cubiertos adecuadamente con los sistemas SIEM actuales que funcionan sobre fuentes tradicionales (firewall, IDS, ...) en una red corporativa*". Desde el estudio hecho por Enterprise Strategy Group a finales de 2012, Olsten [J.13] asevera que el 44% de las empresas consideran Big Data al análisis de la Seguridad, mientras que otro 44% creen que sus requisitos de análisis de seguridad serán Big Data en los dos próximos años.

2.4.1.1 Uso de Hadoop en Sistemas de Detección de Intrusiones

Las plataformas de computación tradicional, como las bases de datos, no escalan de forma adecuada para cubrir las necesidades de los Sistemas de Detección de Intrusiones. Hadoop, es un framework *open-source*, que ofrece una plataforma de almacenamiento distribuida que puede correr en hardware normal (*commodity*). Este framework se usa para cumplir los requisitos de Big Data con respecto al gran Volumen de datos y la alta Velocidad de recogida de flujos desde fuentes heterogéneas. Con el término Hadoop se hace referencia generalmente a varias tecnologías: HDFS, el sistema de archivos distribuidos; Hive, la implementación del DataWareHouse para Hadoop; MapReduce, el paradigma de programación que se usa en Hadoop; Pig, el lenguaje semejante a T-SQL que se usa para realizar consultas en la infraestructura de Hadoop; y un largo etcétera que se describirá en capítulos posteriores [Fou15m].

Suthaharam propone el uso de Hadoop y las tecnologías asociadas al campo de la Detección de Intrusiones, proponiendo 3 Cs para argumentar la adopción del Big Data en el campo: Cardinalidad, número de registros en un instante; Continuidad, (1) flujos de datos de forma continuada, (2) creciendo en los datos con respecto al tiempo; y Complejidad, (1) variedad de datos grande, (2) alta dimensionalidad, (3) alta velocidad de procesamiento. El modelo que plantea Suthaharan está reflejado en la Figura 2.23.

El módulo *User Interaction and Learning System (UILS)* se ocupa de realizar el aprendizaje de los datos, así como de controlar el almacenamiento, permitiendo a los usuarios la interacción con el sistema. El módulo *Network*

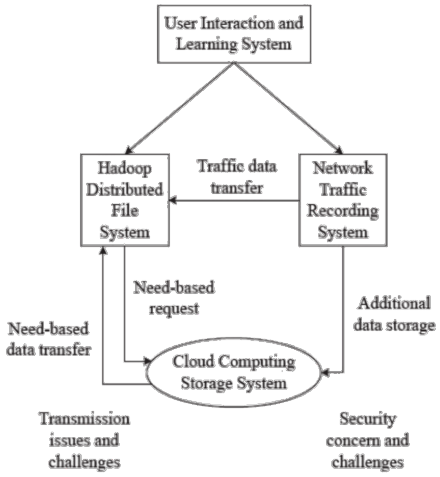


Figura 2.23: Topología sugerida para IDS basados en Big Data.

Traffic Recording System (NTRS) es responsable de la captura del tráfico de red, almacenando el mismo en el *Cloud Computing Storage System (CCSS)* o si es localmente en el sistema sistema HDFS que ofrece Hadoop. Caso de que la información sea requerida inmediatamente se almacena localmente en HDFS, en caso contrario se almacena en el CCSS para ser procesada más adelante. Así mismo, para aplicar técnicas de *Machine Learning* sobre Big Data, Suthaharan indica los tópicos: *multi-domain representation-learning*, *cross-domain representation-learning* y *machine lifelong learning*. El modelo que plantean para la Seguridad se basa en el uso de Internet y las tecnologías *Cloud Computing*[WS14].

Jeong et al. [JHLY12] ofrecen un resumen de las problemáticas encontradas dentro del campo de la Detección de Intrusiones y el Big Data, así como el hecho de que varias tecnologías de Hadoop pueden solucionar estos retos. Lee y Lee [LL13], realizan un experimento haciendo uso de las tecnologías de Hadoop para medir y analizar el tráfico de Internet buscando la detección de ataques DDOS. En el experimento que hicieron fueron capaces de llegar a velocidades máximas de transferencia de 14 Gbps en algunos escenarios, quedándose en 6 Gbps en algunos análisis. Todo ello partiendo de un cluster de 30 máquinas o más, y probando varias opciones con respecto al número de nodos y diversos tamaños de fichero. Si bien, en el estudio que plantearon Lee y Lee sólo se consideró tráfico ya guardado y no monitorización de tráfico en tiempo real. Cheon y Choe [CC13] proponen una arquitectura distribuida de IDS, basada en Snort y Hadoop. Igual que en el caso anterior, se basan en capturas de tráfico ya realizadas en vez

de en tráfico real, y realizan el estudio variando el número de nodos hasta 8. Con los 8 nodos consiguen una mejora de un 424 % en el rendimiento, comparado con una única máquina haciendo el mismo procesamiento. Veetil y Gao [VG13] llevan a cabo un experimento implementando una red Naïve Bayes en un cluster Hadoop, y consiguen realizar el análisis un 37 % más rápido con 6 nodos que con una única máquina, si bien sólo puede llegar a clasificar 434 paquetes por minuto.

2.4.2 Fuentes heterogéneas proporcionando información de Seguridad.

2.4.2.1 Data Fusion.

En el año 2000, Bass [Bas00, Bas99] hizo una gran contribución a la Detección de Intrusiones sugiriendo las técnicas de *Data Fusion* con las más adecuadas para agregar la información de múltiples herramientas de Seguridad. Esta información, de base heterogénea, provenía según su visión de varias fuentes: *network packet sniffers* distribuidos, logs del sistema de ficheros, *traps* y *queries* de SNMP, perfiles de uso de las base de datos, mensajes de sistema y acciones realizadas por el usuario. La idea de Bass era lograr analizar toda la información mediante *Data Fusion* en tiempo real (*online*), dando margen a la minería de datos en modo *off-line* para procesar la enorme cantidad de datos relativos a CyberSeguridad. El concepto de Bass venía de la aplicación del concepto militar OODA: observar, orientar, decidir y actuar; basando todo su trabajo en la idea de múltiples sensores.

Paralelamente, Lan et al. [LCG10] utilizaron *Data Fusion* a través de diversas fuentes heterogéneas, con el objetivo de mejorar la Detección de Intrusiones llevándola a una mejor auto-consciencia. Los autores ya indicaban que las soluciones tradicionales como Firewalls, IDS, y escaners de seguridad, no se integraban bien conjuntamente y que sólo tenían un pequeño conocimiento de los sistemas que protegían. Los autores se basaban en una técnica de *Data Fusion* conocida como D-S (*Dempster-Shafer*), que trata de aplicar probabilidades a las observaciones del sistema en estudio. Los autores indicaban la necesidad de disponer de *Big Velocity* y de *Big Volume*, así como que al recibir información de múltiples fuentes requerían *Big Variety*. Aplicando el concepto con el DataSet de DARPA2000, lograron reducir los mensajes de alerta de 64.481 a 6.164, lo que supone un gran avance en el campo. Se debe tener en cuenta que el fusionar la información de múltiples fuentes, no siempre tiene resultados satisfactorios, ya que

existe un término en el campo que se denomina *catastrophic fusion*, donde el rendimiento del sistema al completo es muy inferior al rendimiento de cada una de las fuentes por separado [Mit12]. .

2.4.2.2 Arquitecturas para datos heterogéneos

En el estudio de Fessy et al. [FBH+10] consideran la Detección de Intrusiones a través de fuentes heterogéneas, dónde múltiples observadores recogen la información de varias fuentes y un analizador global toma la última decisión al respecto. Para poder tomar la decisión final, el analizador global debe trabajar con *Data Fusion* sobre los diferentes flujos, y más teniendo en cuenta posibles ataques distribuidos a la infraestructura al completo, Figura 2.24. Una particularidad de la arquitectura planteada es que los propios analizadores pueden ser heterogéneos, de modo que se pueden combinar sistemas de Detección de Usos Indebidos con sistemas de Detección de Anomalías. Este modelo puede escalar relativamente bien dentro del mundo Big Data, si bien el hecho de que sólo exista un analizador global puede conllevar a tener un cuello de botella.

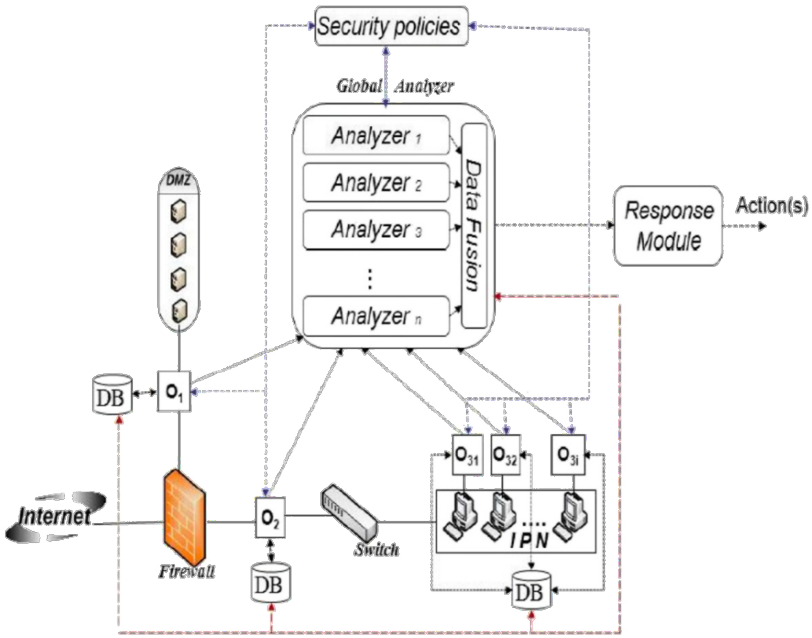


Figura 2.24: Arquitectura propuesta por Fessy et al.

Ganame et al. [GBBS08] extendieron su trabajo previo con un Centro de Operación de Seguridad (SOC), denominado *SOCBox*, y desarrollaron

una versión mejorada del mismo denominada DSOC (*Distributed Security Operation Box*). La arquitectura que proponen permite a una organización crecer y escalar el sistema a través de Internet, proporcionando correlación de eventos incluso en zonas separadas geográficamente, e incluso puede escalarse para soportar a múltiples organizaciones. El motivo principal por el que se extendió la arquitectura era el hecho de que un atacante podía comprometer la red en uno de los sitios, impidiendo que el SOC central tuviera notificación de todos los eventos de seguridad, e incluso presentaron varios ataques tipo *flood* para demostrar esta situación. Así mismo, los autores de esta plataforma coinciden en señalar los beneficios de fusionar múltiples datos, ya que las fuentes heterogéneas permiten un mejor y mayor conocimiento de que lo que está acaeciendo. Los autores hicieron uso de IDMEF (*Intrusion Detection Message Exchange Format*) para el envío de los mensajes, si bien concluyen que es un formato muy pesado y que requiere grandes recursos para procesarlo principalmente con técnicas de correlación de eventos.

El sistema CIDS, *Collaborative Intrusion Detection System*, desarrollado por Bye et al. [BCA10] se fundamenta en tener múltiples participantes (por ejemplo: IDSs, Firewalls, etc.) de equipos que trabajan conjuntamente para garantizar mejor la Seguridad. Esta visión ofrece una nueva imagen de colaboración entre múltiples grupos de trabajo con participantes especializados, que se materializó en un *Framework* denominado CIDF.

Otro modelo propuesto por Bartos y Rehak [BR12] es el denominado DIDS, *Distributed Intrusion Detection System* que trata de superar las limitaciones de los sistemas de detección tradicionales que operan aislados. Los nodos integrantes de esta arquitectura se conocen como sensores y tienen en sí mismos la capacidad de fusionar datos para correlacionar diferentes eventos. Cada uno de los sensores se comunica con el resto con la motivación extra de la redundancia y la mayor recuperación en caso de desastre por un ataque. En sus conclusiones indican que pueden mejorar la precisión en la detección, si bien mantienen constante la tasa de alarmas. En la evaluación de DIDSs, Cai y Wu [CW10] discuten la aproximación de agentes de software, pasando la misma a agentes de host, que se ocupen de monitorizar toda la información relevante para el host en cuestión: sistemas de ficheros, logs y kernel. Cai y Wu también indican como muy positivo correlacionar eventos a través de Internet, al igual que Ganame et al. [GBBS08] y Bartos y Rehak [BR12]. Otros estudios como [ZLK09, MHR11] muestran que la correlación de alertas a través de emplazamientos diferentes geográficamente es una estrategia muy importante contra el Ciber Crimen.

2.4.2.3 **Fuentes de información.**

De cara a poder evaluar la aplicación del Big Data a los sistemas de Detección de Intrusiones, se han realizado numerosas investigaciones para establecer los límites a los que se puede tener acceso y dar respuesta de forma adecuada.

SitaRam et al. [SSZ+13] consideran que los retos a afrontar cuando se evalúa un Network IDS son los proveedores que ofrecen enlaces OC-192 y OC-768, hablando de 9953,28 Mbit/s y de 39.813,12 Mbit/s respectivamente. Los autores anteriores consideran estas capacidades de ancho de banda como una representación clara de flujos Big Data en formato crudo. Sitaram et al. tienen la visión de construir un NIDS capaz de tratar con esos flujos de red por medio de usar herramientas como Hadoop y herramientas de monitorización de redes como PacketPig [G.13]. De acuerdo con los autores de la herramienta anterior, PacketPig, ésta es capaz de realizar *Deep Packet Inspection*, análisis de red en profundidad, e incluso realizar capturas de los paquetes completos. En el paper los autores muestran la efectividad de los algoritmos de cluterización para realizar la clasificación de los paquetes, experimentando sobre el juego de pruebas KDD encontrando K-means como el más representativo y con mejores resultados.

Fuera del ámbito de red, los logs de los hosts han sido tradicionalmente monitorizados, siendo una de los principales fuentes para la Detección de Intrusiones. Una organización puede tener una multitud de ordenadores: estaciones de trabajo, servidores de infraestructura, dispositivos de red (generan logs también), hosts de virtualización, hosts de Cloud Computing, Smartphones, etc.; generando diferentes tipos de ficheros de logs: antivirus, software, firewall, honeypot, web server logs, ftp server logs, email server logs, domain controller logs, web proxy logs, VPN logs, DHCP logs, etc. Siguiendo esta línea de trabajo Yen et al. [YOO+13] desarrollaron un sistema denominado *Beehive* que se ocupa de la gestión y análisis de logs a gran escala, tratando de buscar actividad sospechosa en las redes corporativas, en concreto se enfrentaron a 1,4 billones de mensajes de log, generados de media por día en la compañía EMC. En este paper, además, se hace hincapié en un concepto importante, lo que denominan "datos sucios", por ejemplo: logs con timestamps erróneos o sin referencias de origen por cambios del equipo en la red, etc. En Minería de Datos a este tipo de técnicas para limpieza de los datos se las denomina *Data Cleansing*.

Con toda esta información se puede crear una *Big Host Event Data* que puede cumplir con altos niveles de Volumen, Velocidad y Variedad. Myers

et al. [MGM11] indican que la correlación de eventos en los logs de todas las herramientas corporativas no se realiza de forma frecuente debido a las dificultades y falta de idoneidad de las herramientas tradicionales, pero que se debe abordar desde el punto de vista del Big Data.

2.4.3 **Sistemas SIEM.**

Los sistemas SIEM, *Security Information and Event Management*, son diferentes desde el punto de vista arquitectónico de los sistemas IDS tradicionales, y son el resultado que ofrecen los grandes proveedores de soluciones de seguridad comerciales a la resolución de los problemas que tienen las empresas con respecto a la Seguridad de la Información. El término se acuño en el año 2005 por Mark Nicolett y Amrit Willians, analistas de Garner, para describir cómo estaban convergiendo la gestión de la información de seguridad (SIM, *Security Information Management*) y la gestión de los eventos de Seguridad (SEM, *Security Event Management*). SEM, principalmente se ocupa del análisis en tiempo real como respuesta a un incidente, mientras que SIM se centraba en el almacenamiento a largo plazo para historificación, análisis de tendencias y análisis forense. Según [Res15], actualmente hay 64 productos de SIEM, siendo 6 de ellos *freeware*. Rouse [M.12] ofrece una definición adecuada: los sistemas SIEM ofrecen la posibilidad de ver tendencias y patrones en los datos de Seguridad desde un único punto de vista, incluso teniendo en cuenta que estos datos pueden ser originados por fuentes heterogéneas: eventos de red, dispositivos de usuario, servidores, firewalls, sistemas de antivirus y sistemas de detección y prevención de intrusiones. Aguirre y Alonso [AA12] generalizan las funcionalidades de los sistemas SIEM: agregación de datos de muchas fuentes, monitorización continua de incidentes, correlación de eventos, y notificación de alertas frente a problemas. Kotenko et al. [KPS12] contemplan que los 4 principales componentes de los sistemas SIEM son los siguientes:

1. Filtrado, agregación, abstracción y correlación de eventos.
2. Visualización y razonamiento de los datos.
3. Ayuda a la toma de decisión y aplicación de contramedidas.
4. Modelado de la Seguridad y evaluación de la misma.

Los estándares más usados por los sistemas SIEM son: SCAP [RK11], *Common Base Event* (CBE) [OKS+04] y *Common Information model* (CIM)

[USR+12]. En definitiva, los fabricantes de estos sistemas tratan de que todas las fuentes de información tengan un formato común, de forma que se pueda minimizar el análisis mediante técnicas de *Data Fusion*. Si bien un punto crítico es que en general, todos los sistemas actuales trabajan sobre un modelo de base de datos tradicional, que en general suele acabar sobrecargada. Para superar esta limitación Kotento et al. [KPS12] recomiendan hacer uso de una aproximación híbrida, donde la base de datos tradicional se use conjuntamente con bases de datos basadas en XML y con almacenamiento de tripletas [1]. Kotenko y Chechulin [KC12] proponen un framework para el modelado de ataques en los sistemas SIEM denominado AMSEC (*Attack Modeling and Security Evaluation Component*) que trata de solventar tanto las vulnerabilidades conocidas como las desconocidas.

Metzger et al. [MHR11] tratan de dar un paso más ofreciendo la unión de los sistemas SIEM con las técnicas formalizadas de Gestión de Incidentes. La idea es que el sistema al completo reaccione de forma automática o manual a través de un equipo CSIRT (*Computer Security Incident Response Team*), de forma que se pueda afrontar el problemas de las *botnets*, *email spam* y otro tipo de ataques semejantes. La propuesta incluye dos fuentes adicionales en los sistemas SIEM: Reporte Manual y el servicio DFN-CERT. El Reporte Manual permite a las organizaciones recoger la información de sus usuarios relativa a problemas de seguridad, de forma que el sistema SIEM pueda procesar también esta información ya contrastada con el operador humano. El sistema DFN-CERT es un servicio que se desplegaría a nivel internacional en el que se reporta automáticamente el comportamiento y los metadatos de los incidentes, con la mejora bidireccional de la correlación de eventos.

Los sistemas SIEM en la actualidad se ven desbordados, tanto para la recogida de información, como para el procesamiento de la misma en búsqueda de intrusiones. Así mismo, suelen generar una cantidad de alertas inusitada que no reflejan amenazas reales, obviando por las técnicas de correlación que utilizan de ataques elaborados, que quedan camuflados por la ingente cantidad de datos a tener en cuenta. Como queda demostrado, este sector adolece de un cambio, como indica el producto de Splunk [Spl15], en el que las bases de datos tradicionales den paso a las nuevas tecnologías de Big Data.

[1] Un almacenamiento basado en tripletas es una base de datos para almacenamiento y recuperación de metadatos tipo RDF, usados generalmente en Ontologías para establecer relaciones entre las entidades.

Parte I

ESIDE-BIDS. Big Heterogeneus Data Based Intrusion Detection Framework.

«Sciencia potentia est (Traducción: El conocimiento es poder).»

Sir Francis Bacon (1561 - 1626)

Capítulo

3

Recogida de Evidencias

El primer problema que se debe abordar a la hora de innovar en el área de la Detección de Intrusiones es la generación de corpus de evidencias suficientemente significativo como para poder reproducir los distintos tipos de ataques que existen, teniendo en cuenta que el grueso del tráfico y eventos de hosts que se producen habitualmente no contienen ataques, haciendo que éstos últimos sean poco relevantes estadísticamente. El presente capítulo se ocupa de explicar las tecnologías de recogida de la información que se van a utilizar en la presente tesis, este capítulo continuará en la parte experimental con el sistema de recolección implementado.

3.1 DataSets públicos en el área de la Detección de Intrusiones.

Muchos autores han discutido los problemas que existen con los *DataSets* públicos que aplican al área de la Detección de Intrusiones. Por ejemplo, Sommer y Paxson [SP10] hacen un excelente resumen sobre las razones por las que esto es un problema significativo. El estado de la cyber-seguridad ha evolucionado enormemente en los últimos años y consideran que las publicaciones que se realizan actualmente sobre *DataSets* obsoletos no deberían ser consideradas como relevantes. Aunque, como es normal, muchas organizaciones no desean proporcionar datos reales, incluso aunque se anoni-

micen los mismos [CWM+07]. Los datasets más famosos y utilizados dentro del ámbito son el de DARPA y el denominado KDD Cup, si bien ambos son de hace más de 10 años y tienen varios problemas ya conocidos como se discute por McHugh [McH00] y Mahoney y Chan [MC03]. Algunos de los *DataSets* más utilizados comúnmente están en la tabla 3.1.

Cuadro 3.1: Resumen de los DataSets más populares en el área de la Detección de Intrusiones.

Fuente de Datos	Nombre del DataSet	Abreviatura
Network Traffic	DARPA 1998 TCPDump Files	DARPA98
	DARPA 1999 TCPDump Files	DARPA99
	KDD99 DataSet	KDD99
	10% KDD99 DataSet	KDD99-10
	Internet Exploration Shootout DataSet	IES
User Behaviour	Unix User Dataset	UNIXDS
System Calls	DARPA 1998 BSM Files	BSM 98
	DARPA 1999 BSM Files	BSM 99
	University of New Mexico Dataset	UNM

La mayor pega que adolecen estos juegos de datos es que no cumplen con la Veracidad desde la perspectiva de *Big Data*, por lo que no son relevantes como consecuencia de la baja calidad de datos. Así mismo, debido a la baja Veracidad, estos *DataSets* carecen de Valor, reduciendo aún más la relevancia de los mismos en estudios serios.

Hay más juegos de datos como ISCX [SSTG12], MAWI [FBAF10], NSA Data Capture [Poi15] o Internet Storm Center (que incluye dshield.org) [Cen15]. Si bien, estos últimos pese a ser más actuales no son usados con frecuencia en comparación con el de DARPA o el KDD, incluso en los estudios más recientes.

Otras aproximaciones pasan por la creación de Honeypots, como por ejemplo en Song. et al [STO+11], que incluye incluso tráfico normal generado por ellos mismos. La generación de tráfico en entornos de laboratorio no tiene porqué ser más realista, si bien es la técnica más recomendada actualmente para llevar una investigación significativa, esto es: generar un entorno de trabajo con tráfico normal y lanzar ataques dirigidos de la complejidad deseada. Si bien, se ha trabajado en la generación de un entorno apropiado para la creación de los juegos de ensayo necesarios, en la sección 5.1 se analizará más en detalle esta base de datos por medio de tecnologías Big Data, como punto de partida a la experimentación de la presente tesis

doctoral.

3.2 Sistemas SIEM.

La tecnología SIEM se ocupa de agregar los eventos producidos por dispositivos de seguridad, infraestructuras de red, sistemas y aplicaciones; siendo su principal fuente de datos la información almacenada en *logs*. Este tipo de sistemas procesa información heterogénea de diversos tipos, incluyendo flujos de agregados de red, así como datos en crudo de los paquetes que fluyen en la capa de comunicaciones. Toda la información que recogen es contrastada con información contextual de usuarios, procesos, recursos, amenazas y vulnerabilidades. Los datos se normalizan haciendo que aunque provengan de diversas fuentes puedan ser correlacionados y analizados para diferentes propósitos. El objetivo y funcionalidad principal de la tecnología es el procesamiento en tiempo real de los eventos de seguridad, así como el proporcionar herramientas a los consultores de seguridad para analizar los datos históricos, entender lo acontecido y generar reportes adecuados para la resolución de incidentes.

Durante el año 2014 la demanda de este tipo de sistemas ha sido creciente, aumentando el número de clientes en más de un 24% con respecto al año anterior y pasando de una tasa de mercado de 1,5 billones de dólares a 1,69 billones. El mercado en el que compiten es clasificado como maduro y muy competitivo, haciendo que los clientes pasen de una solución a otra en cuanto se encuentran con problemas en despliegues grandes. El fallo principal de este tipo de sistemas está en la detección de brechas originadas por *Zero Days*, dándose la situación de que en el 92% de las compañías que han sufrido este tipo de ataques con sistemas SIEM desplegados no han sido conscientes hasta tiempo después. Según Gartner se indica en el reporte que las áreas de trabajo más fuertes que tienen que cubrir los productos actuales son: la gestión inteligente de amenazas, la mejora del perfilado del comportamiento y la mejora de los motores de análisis.

Las tecnologías SIEM se componen de dos partes y ambas deben ser proporcionadas para entrar en la definición:

- SIM: Gestión, análisis y reporte de logs.
- SEM: Monitorización y gestión de incidentes en tiempo real de eventos relacionados con la seguridad provenientes de: redes, dispositivos de seguridad, sistemas y aplicaciones.

Los casos de uso principales a los que se destina esta tecnología son los siguientes:

- Gestión de incidentes y/o amenazas: Monitorización y reporte en tiempo real de actividad del usuario y aplicaciones, así como del acceso a datos.
- Conformidad: Recogida de información de múltiples fuentes, gestionando las mismas y permitiendo incluir la información de ellas en reportes asociados.
- Despliegues mixtos de los dos casos de uso anteriores.

Figura 3.1: Cuadrante Mágico de sistemas SIEM.

En la Figura 3.1 se muestra la resolución del año 2015 de la consultora Gartner [KMK15] para este tipo de sistemas. La clasificación se realiza en base a los siguientes criterios:

- Líderes: Se clasifican como líderes aquellos vendedores que proporcionan productos que tienen un fuerte cumplimiento con las necesidades del mercado actual, que obtienen un beneficio sustancial dentro del mercado de SIEMs y tienen asignada una viabilidad muy alta a futuro. Estos SIEMs son los que mejor se posicionan con las necesidades futuras y disponen de una tasa de mercado elevada.

- Desafiadores: Aquellos fabricantes que tienen múltiples productos o servicios, una base de clientes de tamaño modesto y productos que reúnen un subconjunto de los requisitos del mercado general. Los productos en este cuadrante tienen fuertes capacidades de ejecución: recursos financieros, comerciales, y presencia de marca; pero no disponen de todos los requisitos necesarios para pertenecer al cuadrante de líderes.

- Visionarios: En este cuadrante se incluyen aquellos vendedores que proporcionan productos que cumplen con los requisitos generales del mercado de SIEMs, pero disponen de una capacidad menor de ejecución a nivel de mercado que los líderes.

- Vendedores Nicho: Se compone de aquellos vendedores que proporcionan tecnologías que se ajustan a un caso de uso concreto de los SIEM, centrándose en un subsegmento del mercado. Generalmente los proveedores en este cuadrante disponen de menor solvencia, planes de crecimiento modestos y mejor capacidad de ejecución a futuro.

Dentro del mercado, el 60% se lo llevan los grandes fabricantes: HP, IBM, Intel y Splunk. Todos ellos llegan a los factores de escalabilidad más altos, que según Gartner pueden fijarse en los siguientes despliegues:

- Pequeño: menos de 300 fuentes, menos de 1.500 eventos por segundo (EPS) y un almacenamiento histórico de 800 GB o menos.

- Mediano: entre 400 y 800 fuentes, entre 2.000 y 7.000 EPS, con un almacenamiento entre 4 y 7 TB.

- Grande: más de 900 fuentes, más de 15.000 EPS y un almacenamiento superior a 10 TB.

- Gigante: varios miles de fuentes, más de 25.000 EPS y más de 50 TB de almacenamiento

3.2.1 Grandes fabricantes: HP, IBM, Intel y Splunk.

Una de las soluciones más conocidas en el mercado es **ArcSight**, el SIEM de **HP**, que se compone de un gestor conocido como ESM (*Enterprise Security Manager*) para grandes despliegues, despliegues sólo de la parte SEM, y el ArcSight Express, que es un appliance ofrecido a los segmentos medianos. HP proporciona módulos adicionales más especializados, entre otros: *Application View*, que proporciona información de ejecución de las aplicaciones corriendo basado en la tecnología HP Fortify; *User Behavioral Analytics*, que se basa en la tecnología de **Securonix** para realizar el análisis de los usuarios. Se licencia en base al consumo de GB que se realiza por día.

La solución de **IBM** se compone de **QRadar**, el gestor de logs, el gestor de vulnerabilidades, el gestor de riesgos, los recolectores de datos QFlow y VFlow, y los sistemas forenses integrados. Se puede desplegar como solución en *appliance*, en entorno virtual e incluso en arquitecturas IaaS. Las tecnologías denominadas como QRadar permiten la recogida de información y procesamiento de logs, datos NetFlow, DPI, captura de paquetes de red completa y análisis del comportamiento. Los usuarios citan como distintivo su capacidad de procesamiento, su escalabilidad, las reglas de correlación, la estabilidad y la calidad general de la solución.

Intel Security se basa en el gestor empresarial de seguridad de **McAfee**, que combina funcionalidades SIM y SEM, estando disponible como *appliance* físico, virtual o basado en software. Tiene tres componentes principales: el gestor de securidad corporativa (ESM), el receptor de eventos (ERC) y el gestor de logs (ELM), que se pueden desplegar conjuntamente en la misma máquina y ser distribuidos. Sus funcionalidades pueden ser ampliadas mediante productos especializados como el sistema de correlación avanzada (ACE), el monitor de bases de datos (DEM), el monitor de aplicaciones (ADM) y el gestor inteligente de amenazas (GTI). Además agrega como fuentes de conocimiento la información proporcionada por McAfee. Entre sus fortalezas, aparte de los despliegues gigantes a los que puede dar solución, es la cobertura de soluciones tipo SCADA para el mundo industrial.

Splunk proporciona un sistema basado en la nube que ofrece funciones de búsqueda, alerta, correlación en tiempo real y un lenguaje de búsqueda que posibilita la visualización de más de 100 comandos estadísticos. Los clientes indican que las mayores fortalezas son la visualización, las capa-

cidades de análisis estadístico y predictivo del comportamiento, así como la detección de acceso a datos sensibles por parte de usuarios no autorizados. Splunk soporta *feeds* de fuentes externas de soluciones comerciales y *open-source*.

3.2.2 **Soluciones basadas en open-source: AlienVault.**

La solución **AlienVault Unified Security Management (USM)** proporciona sistemas de SIEM, de evaluación de vulnerabilidades (VA), descubrimiento de recursos, sistemas de detección de intrusiones de Host y de Red (HIDS, NIDS) y sistemas de integridad de ficheros (FIM). El SIEM de AlienVault proporciona un entorno centralizado para la gestión y monitorización de todos los componentes. AlienVault está compuesta de componentes *open-source*:

- OpenVA.
- Snort, Suricata como NIDS.
- OSSEC como HIDS y FIM.

AlienVault ofrece un SIEM gratuito denominado OSSIM, que siendo libre, ofrece una versión limitada de su solución comercial completa. AlienVault USM extiende OSSIM con mejoras en la escalabilidad, gestión de logs, reporte y administración consolidados, y federación con MSSPs. El vendedor además ofrece la integración con el sistema de tratamiento inteligente de amenazas, que incluye reputación web y de tráfico IP. Los laboratorios de la firma ofrecen a sus clientes comerciales actualizaciones de: firmas, vulnerabilidades, reglas de correlación, reportes avanzados y contenido para la resolución y respuesta en caso de incidentes.

La solución está disponible como un *appliance*, como software y como imagen virtual, pudiendo ser desplegada en Amazon EC2 (*Amazon Elastic Compute Cloud*). Los sensores, los sistemas de logeo y los componentes del servidor pueden ser desplegados en un único sistema combinados o como servidores separados en capas horizontales y verticales para escalar debidamente y ajustarse a los entornos de clientes finales. A finales de 2014, AlienVault creó un nuevo producto para poder instalarse dentro de Amazon AWS (*Amazon Web Services*).

El mercado al que se destina la solución está centrado en empresas medianas o pequeñas, con poco personal dedicado a la Seguridad de la Información, pocos programas destinados a la protección de la seguridad, bajo

costo de implantación, simplicidad y recogida de información de múltiples fuentes heterogéneas. El licenciamiento de la solución se basa en el número de nodos utilizados, más que en el número de eventos por segundo (EPS) que usan las grandes distribuciones. Entre sus puntos fuertes los usuarios indican que la generación de reglas de correlación propias es más potente que la que ofrecen sus competidores, si bien al estar basada en open-source adolece de problemas de cuando en cuando.

La solución AlienVault se fundamenta en la comunidad *open-source*, mediante una distribución libre que se denomina OSSIM y una solución de pago donde se despliega toda la tecnología para unificar en una consola la información de todos los sistemas de IDS/IPS. Las fuentes de información de AlienVault USM, la versión de pago, al igual que en la versión *open-source* son:

- *Network Intrusion Detection System (NIDS)*: Recoger amenazas sobre equipos vulnerables mediante detección de anomalías basadas en firmas y tecnologías de análisis de protocolos. Permite identificar los últimos ataques, infecciones de malware, sistemas comprometidos, violaciones a la política de seguridad y otras posibles brechas mediante análisis del tráfico de red.

- *Host Based Intrusion Detection System (HIDS)* y *File Integrity Monitoring (FIM)*. Se analiza la actividad en los sistemas y el estado de la configuración de los mismos, llevando trazabilidad de la actividad de usuario y sus accesos. Mediante el sistema FIM puede detectar sistemas comprometidos, modificaciones a los ficheros críticos (como claves de registro, */etc/passwd*, etc.), rootkits comunes y procesos maliciosos.

La diferencia más importante entre sus versiones de pago y la versión *open-source* viene de la mano de las dos siguientes características:

- El motor de correlación avanzada con más de 2.000 directivas para alertar sobre las amenazas más importantes, que la versión libre no lo tiene, permitiendo tasas de transferencia de datos y análisis mucho menores que la versión comercial.

- La actualización de contenidos con las firmas más representativas desarrolladas por el propio equipo de AlienVault, que en la versión libre no tiene actualización constante, ni consultoría especializada.

- Las plantillas de reporte que tiene cada versión, siendo 3 en el caso de OSSIM y más de 150 en el caso de AlienVault USM.
- El número de usuarios que permite cada sistemas SIEM, siendo OSSIM monousuario.

3.2.2.1 **OSSIM.**

En la presente tesis se ha estudiado la versión *open-source*, denominada OSSIM [Ali15], como punto de partida a la recogida de evidencias, que proporciona al usuario con un sistema SIEM completo, rico en características y que realiza la recolección de evidencias, normalización de las mismas y la correlación. En la configuración del entorno, lo primero que se pregunta es el tipo de instalación a realizar, si se está instalando un sensor o el manager de la infraestructura, como se puede ver en la figura 3.2.

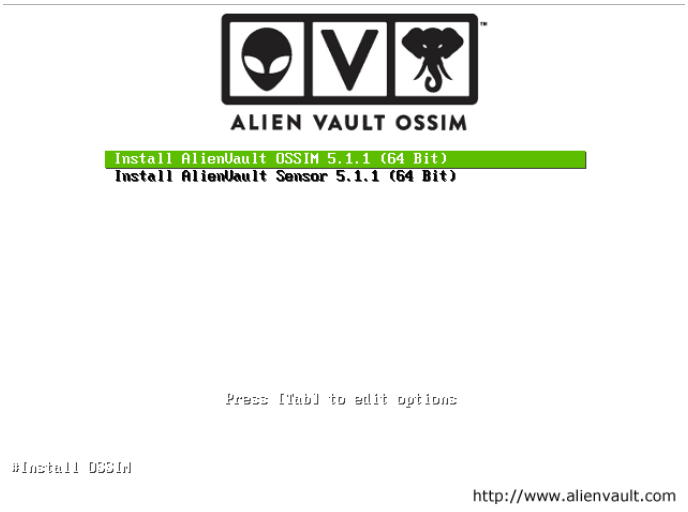


Figura 3.2: OSSIM - Instalación, manager o sensor.

Los siguientes pasos que pide la instalación del *ossim-manager* son (figura 3.3): la configuración de las redes de monitorización, mantenimiento y recogida de logs; el descubrimiento de los servicios activos, tanto dispositivos de red como dispositivos de host, para finalmente comenzar la monitorización. Con la propia interfaz web se pueden desplegar todos los agentes que se estimen necesario, agrupando las fuentes de información de forma coherente con la estructura de la empresa a monitorizar. En esta pantalla además se pueden configurar tanto los agentes de OSSIM instalados en máquinas físicas, virtuales o *appliances*, como los agentes residentes que

se instalarán usando las cuentas de usuario administrado de Dominio o de otros sistemas (UNIX, Linux, Mac OS, etc.).

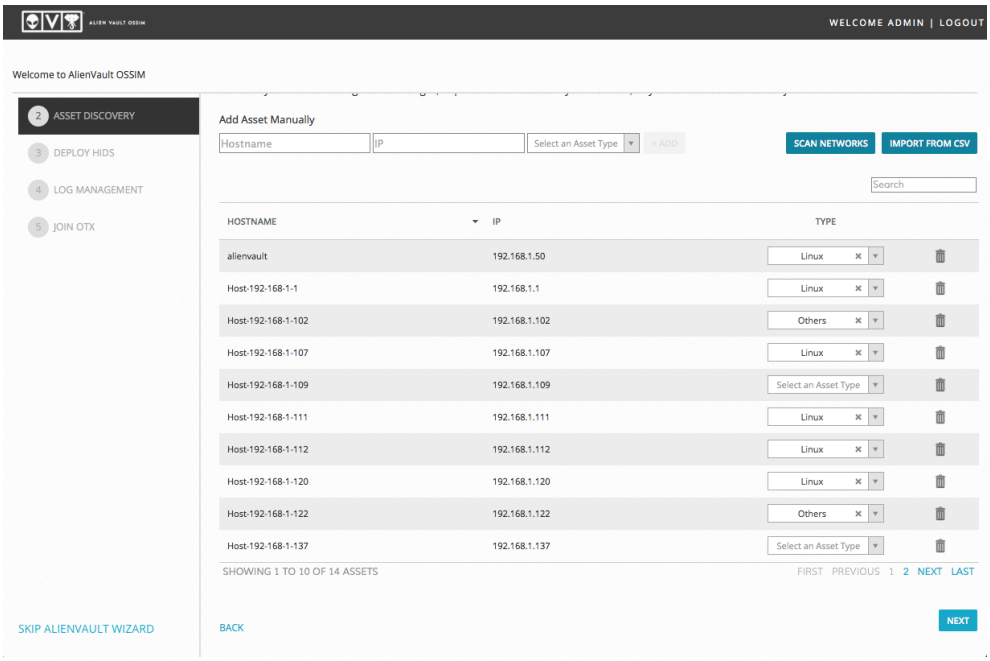


Figura 3.3: OSSIM - Configuración.

Finalmente, cuando se completa el despliegue, se dispone de una herramienta de cuadro de mando que resume los principales eventos por criticidad, permitiendo controlar la infraestructura al completo desde un punto centralizado (figura 3.4).

Tal y como se ha indicado, el modelo presentado en esta tesis se basa en la aplicación de *Big Data* a los sistemas de Seguridad Integrales (SIEM), y debido a las arquitecturas estándar en ellos centralizando el conocimiento en una base de datos relacional (RDBMS), se ha trabajado en la recogida de evidencias directamente desde las fuentes más importantes de estos sistemas: la recogida de información de *host* mediante OSSEC, sección 3.3; y la recogida de información de red mediante Snort, sección 3.4.

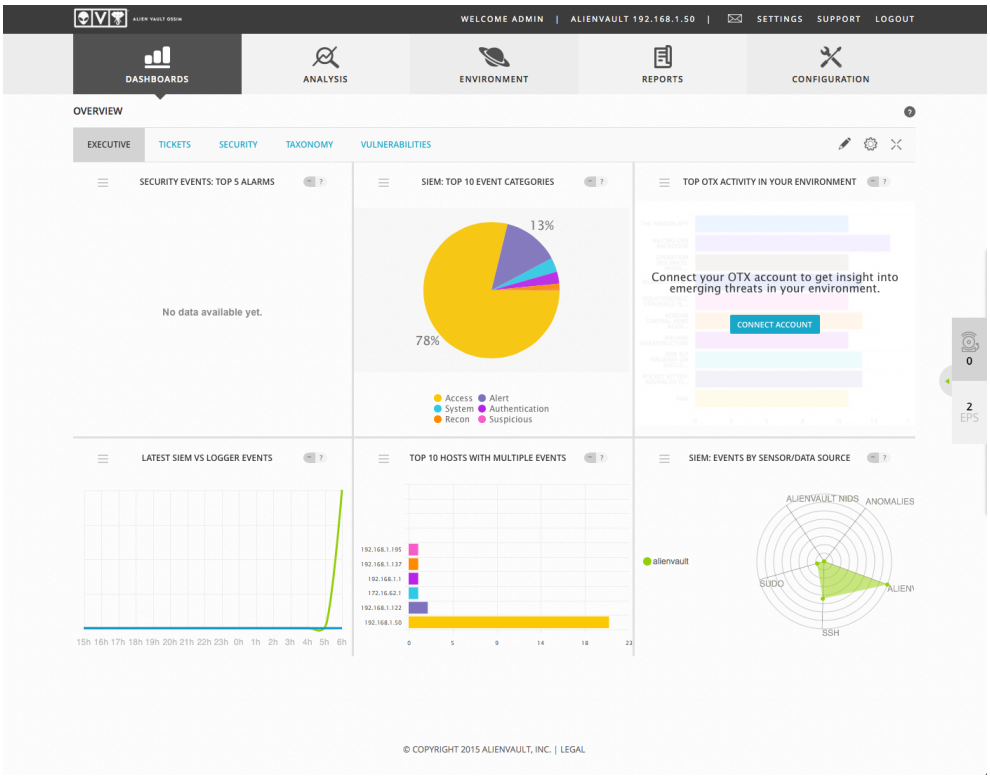


Figura 3.4: OSSIM - Cuadro de mando.

3.3 OSSEC.

OSSEC [Mic15] es una plataforma para monitorizar y controlar los sistemas, mezclando funcionalidades de HIDS (*host-based intrusion detection system*) con la monitorización de logs y la gestión de incidentes/eventos de seguridad (SIEM). Es un sistema multiplataforma que puede operar en varios sistemas operativos: Linux, Solaris, Windows y Mac OS X. El sistema esta preparado para acometer recogida de información, generalmente proveniente de logs, procesamiento y alerta en tiempo real, categorizando por nivel de severidad los eventos ocurridos.

Si bien la arquitectura de OSSEC permite hacer un despliegue local, lo más acertado de este tipo de solución es un despliegue centralizado, en el que un servidor (virtual o físico) se ocupa de hacer de gestor de la solución, dejando agentes (clientes) residentes en los sistemas operativos remotos a monitorizar. Los remotos pueden ser basados en agentes residentes o bien simplemente el gestor se ocupará de recabar los logs de sistemas externos. Este último procedimiento abre la puerta para que OSSEC pueda recoger logs de monitorización de sistemas de red, como *routers* y/o *firewalls*. La figura 3.5 muestra los componentes principales de la arquitectura OSSEC.

Figura 3.5: OSSIM - Configuración.

Entre las características más destacadas de este sistema SIEM, se encuentran las siguientes:

- Comprobación de la integridad de ficheros. Esta funcionalidad, FIM

(*File Integrity Monitoring*), se ocupa de reconocer y alertar cuando algún proceso cambia los sistemas de ficheros que monitoriza de alguna forma, bien sea por un mal uso no intencionado (errores del administrador o del usuario) o bien sea por un ataque, se notificará del suceso escalando a niveles superiores por si requiere actuación posterior. Dispone de una funcionalidad semejante para sistemas operativos Windows, que denominan *Registry Integrity Checking*.

- Monitorización de logs. [1]
- Detección de *rootkits*.
- Respuesta activa. Esta última permite ejecutar secuencias de comandos tras la detección de un suceso anómalo, haciendo que la respuesta del sistema evite que se propague el malware o que el atacante pueda continuar conectado o pasar a otro host vulnerable.

Los dispositivos soportados por OSSEC son los siguientes:

- Sistemas Operativos:
  - GNU/Linux (todas las distribuciones)
  - Windows XP, 2003, Vista, 2008, 2012
  - VMWare ESX 3.0,3.5
  - FreeBSD (todas las versiones actuales)
  - OpenBSD (todas las versiones actuales)
  - NetBSD (todas las versiones actuales)
  - Solaris 2.7, 2.8, 2.9 y 10
  - AIX 5.2 y 5.3
  - Mac OS X 10.x
  - HP-UX 11
- Dispositivos soportados vía *syslog*:
  - Cisco PIX, ASA and FWSM (todas las versiones)
  - Cisco IOS routers (todas las versiones)

[1] En algunos foros se entiende esta tecnología como LIDS, *Log Based Intrusion Detection System*, sin embargo no deja de ser parte de HIDS.

  - Juniper Netscreen (todas las versiones)
  - SonicWall firewall (todas las versiones)
  - Checkpoint firewall (todas las versiones)
  - Cisco IOS IDS/IPS module (todas las versiones)
  - Sourcefire (Snort) IDS/IPS (todas las versiones)
  - Dragon NIDS (todas las versiones)
  - Checkpoint Smart Defense (todas las versiones)
  - McAfee VirusScan Enterprise (todas las versiones)
  - Bluecoat proxy (todas las versiones)
  - Cisco VPN concentrators (todas las versiones)
  - VMWare ESXi 4.x

- Dispositivos soportados sin agente:
  - Cisco PIX, ASA y FWSM (todas las versiones)
  - Cisco IOS routers (todas las versiones)
  - Juniper Netscreen (todas las versiones)
  - SonicWall firewall (todas las versiones)
  - Checkpoint firewall (todas las versiones)

El sistema en sí mismo se puede entender como un detector de usos indebidos que tiene cientos de reglas en su interior, almacenadas en *$INSTALL_DIR/rules/*.xml*. Para evitar que los servicios centrales o los agentes sean el punto de entrada debido a una vulnerabilidad, todos ellos corren dentro de una jaula o *chroot* con unos privilegios de usuario muy limitados.

3.3.1 **Reglas y flujo interno.**

El flujo interno dentro de la infrastructura sigue una estructura bastante directa en el servidor central: los demonios de recolección de logs, recogen la información de los agentes; los logs recogidos se decodifican para poder poner todos los eventos en un formato normalizado y explotable; después los eventos normalizados se pasan por una serie de expresiones regulares que son las denominadas *reglas* y al proceso en OSSEC se le llama "análisis"; finalmente, caso de que se superen los umbrales se realizará tanto el envío

de las alertas por correo, como la ejecución de los comandos proactivos para aislar o prevenir la intrusión. Toda la comunicación entre los agentes y el servicio de recolección se realiza por medio de UDP, en el puerto 1514, y está codificada criptográficamente con claves pre-compartidas y el algoritmo *blowfish*. El flujo de OSSEC puede verse en la figura 3.6.

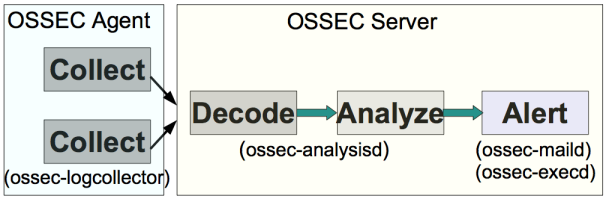


Figura 3.6: OSSEC - Flujo de Reglas.

El proceso de decodificación tiene dos partes, la primera una predecodificación se ocupa de la extracción de información común (nombre del host, nombre del programa, *timestamp*, etc.). La segunda fase se hará la decodificación del resto de la información que se genera por evento, siendo una estructura en árbol de cada campo dentro de OSSEC, pasando finalmente a aplicar las reglas sobre la información para ver si tiene que lanzarse una alerta o no [1]. En el código 3.1 se pueden ver algunas reglas de OSSEC, donde es importante señalar el nivel de criticidad que tienen asignado por conocimiento experto (campo *level*). Estás reglas se aplican sobre los eventos haciendo uso de expresiones regulares para la extracción de contenido. En la figura 3.7 se presenta la interfaz web que ofrece el proyecto para la gestión de las alertas.

Código 3.1: Ejemplos de reglas de OSSEC

```
...
<rule id="30101" level="0">
<if_sid>30100</if_sid>
<match>^[error] </match>
<description>Apache error messages grouped.</description>
</rule>
 ...
 <rule id="2501" level="5">
 <match>FAILED LOGIN |authentication failure|</match>
 <match>Authentication failed for|invalid password for|</match>
 <match>LOGIN FAILURE|auth failure: |authentication error|</match>
 <match>authinternal failed|Failed to authorize|</match>
 <match>Wrong password given for|login failed|Auth: Login incorrect|</match>
 <match>Failed to authenticate user</match>
```

[1] Es importante señalar las reglas se pueden anidar en forma de árbol, haciendo un filtrado sobre las hojas finales caso de cumplir los requisitos superiores.

```
<group>authentication_failed,</group>
<description>User authentication failure.</description>
</rule>
...
```

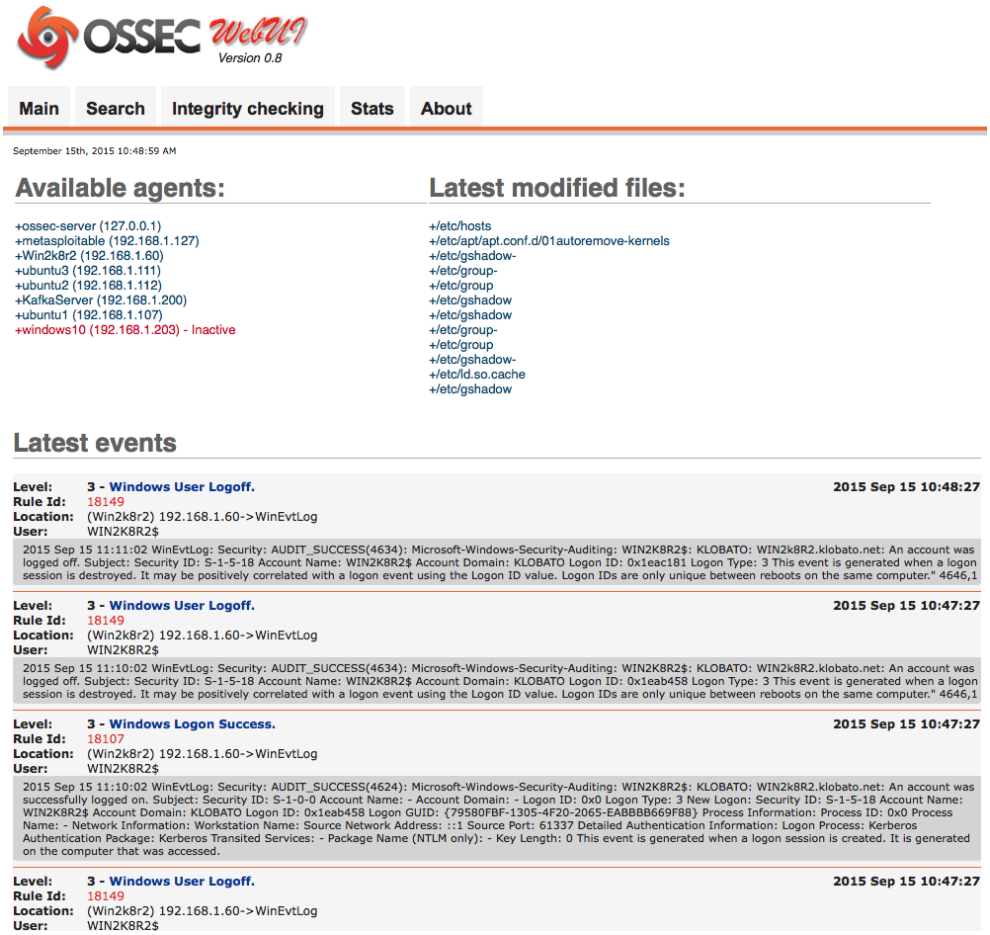


Figura 3.7: OSSEC - Interfaz Web.

3.4 **Snort.**

Snort [aiA15] es un proyecto *open-source* de tipo NIDS/NIPS (*Network Intrusion Detection System / Network Intrusion Prevention System*) que funciona sobre versiones de sistema operativo UNIX-like, aunque también hay algunos *ports* a Windows. El creador del software originalmente fue Martin Roesch en 1998, liberando el código con licencia GPL (*General Public License*). Es uno de los proyectos más conocidos y reconocidos en el mundo de la Seguridad de la Información.

Básicamente la funcionalidad principal de Snort es estar a la escucha de los paquetes de red en uno o en varios segmentos de la misma. En su configuración más simple se pone(n) la(s) tarjeta(s) de red en modo promiscuo recibiendo todo el tráfico que pasa por la red (puede requerir redirección del tráfico en segmentos controlados por routers gestionados o con VLANs / MPLS). Los paquetes que se reciben, tras pasar una serie de preprocesadores y procesadores, finalmente se tratarán de comparar con las reglas que tiene Snort, es decir, no deja de ser un sistema de Detección de Usos Indebidos.

Los componentes básicos del flujo de procesamiento de datagramas de red en Snort son los siguientes:

- Decodificador de Paquetes de Red. Este componente simplemente recoge la información de diferentes tipos de interfaces de red (Ethernet, SLIP, PPP, etc.) y prepara los paquetes para ser preprocesados o enviados al motor de detección.

- Preprocesadores. Son componentes que se se usan para agregar o modificar los paquetes de datos antes del motor de detección. Algunos de los preprocesadores incluso buscan anomalías en los formatos de los mensajes, revisando sus cabeceras. Es muy conocido el módulo *frag3*, que se ocupa de la fragmentación de paquetes muy grandes. Otros preprocesadores de información: *Session*, *Stream*, *RPC Decode*, *HTTP Inspect*, *SMPT*, *POP*, *IMAP*, *FTP*, *SSH*, *DNS*, *ARP*, *SIP*, *Modbus*, etc. En estos módulos, además, se dispone de filtrado por eventos, supresión en función de alguna regla o incluso filtrado por número de eventos por segundo (*flooding*).

- Motor de detección. Esta es realmente la parte crítica de Snort, ya que dependiendo de lo rápido que funcione se establecerá el flujo máximo de datos que se puede tratar. Un ejemplo de reglas que usa el motor de detección se puede ver en el código 3.2.

- Logeado y sistema de alertas.
- Módulos de salida. Los módulos de salida toman la información resultante de sistema de logeado y de alertas anterior, pudiendo escribir la misma en los siguientes formatos: binario tipo *tcpdump*, volcado a log extendido, volcado a log de sistema (*syslog*), ficheros CSV, formato binario tipo *unified2* y otras salidas para sistemas externos como SIEMs.
- Se dispone además de módulos especiales de respuesta activa, es decir, módulos que se ocupan de filtrar el tráfico cuando se funciona en modo Prevención de Intrusiones o modo "*inline*".

Código 3.2: Ejemplo de Reglas de Snort - Servidor Apache

```
alert tcp \$EXTERNAL\_NET any -> \$HTTP\_SERVERS \$HTTP\_PORTS (msg:"SERVER-APACHE HP
     Operations Dashboard Apache Tomcat default admin account access attempt";
     flow:to\_server,established; content:"/manager/"; fast\_pattern:only; http\_uri;
     content:"Authorization: Basic"; nocase; http\_header;
     content:"ajJkZXBsb3llcjpqMmRlcGxveWVy"; distance:0; metadata:service http;
     reference:bugtraq,36258; reference:cve,2009-3098; reference:cve,2009-4188;
     classtype:attempted-admin; sid:24306; rev:3;)
alert tcp \$EXTERNAL\_NET any -> \$HOME\_NET \$HTTP\_PORTS (msg:"SERVER-APACHE Apache Struts
     remote code execution attempt - POST parameter"; flow:to\_server,established;
     content:".action"; fast\_pattern:only; http\_uri; content:"new"; nocase;
     http\_client\_body; pcre:"/new(\s|\%20)(java|org)/iP"; metadata:policy balanced-ips
     drop, policy security-ips drop, service http; reference:cve,2012-0391;
     reference:url,issues.apache.org/jira/browse/WW-3668; classtype:attempted-admin;
     sid:23631; rev:3;)
```

Actualmente, la versión está en la rama 2.9 y ha cambiado la forma en la que se capturan los paquetes de datos de red. Al principio se hacia uso de la librería *libpcap* directamente, si bien, ahora se ha dispuesto una librería nueva denominada DAQ, lanzada en forma experimental a mediados de 2010, pero no usada hasta las últimas versiones con su funcionalidad plena. Con este nuevo sistema, que sigue haciendo uso de *libpcap* internamente para la captura de paquetes, pero que al añadir *libdnet* para el resto de manipulaciones a nivel de red se ha abierto la configuración simplificada de sistemas NIPS. Esta configuración, si bien ha estaba presente en el pasado, con el modo "*inline*", viene a ser simplificada, más configurable y considerablemente más versátil.

3.4.1 Complementos: Barnyard2 y Pulledpork.

Como se ha comentado, Snort, no deja de ser un sistema de detección de usos indebidos, es por ello que continuamente precisa de actualizar su co-

nocimiento, recogiendo información de expertos externos. Hay muchas comunidades que desarrollan este tipo de reglas, si bien las más conocidas vienen de la mano de la propia comunidad y de los expertos que ofrecen el servicio: la empresa SourceFire inicialmente, que fue adquirida por Cisco. Para lidiar con esta tarea tediosa, se dispone del proyecto **Pulledpork** [shi15], que permite la automatización de la descarga de reglas, así como el parseado de las mismas, la modificación de estado del sistema y la actualización de las reglas para Snort.

Por otra parte, una de las limitaciones más conocidas de Snort viene de la mano de la forma en la que se generan los mensajes de alertas. En redes donde el tráfico es muy intenso suele haber problemas de rendimiento con Snort si no se configura adecuadamente. Snort no procesa el siguiente paquete hasta que termina de escribir la alerta por los *plugins* de salida. Al tener que procesar un gran volumen de información el rendimiento de Snort se ve afectado si tiene que escribir las alertas a una base de datos, ya que conlleva una latencia alta. Para evitar esta situación, la solución viene de la mano de escribir a un fichero local en formato binario rápidamente, evitando latencias al escribir las alertas y por ello un DoS (*Denial Of Service*).

El proyecto Barnyard2 [fir15] se ocupa de el traslado de la información de las alertas generadas en el formato binario a una base de datos externa u otros sistemas (PRELUDE, BRO, OSSIM, etc.). El formato que se usa para salvar el binario de la información de Snort es ***unified2***. Barnyard2 puede funcionar en forma de lotes, en base a los ficheros que se le indiquen de entrada, o de forma continua: procesando ficheros tan pronto como se generen, pudiendo poner un testigo, denominado *waldo* con el *timestamp* de la última alerta/log procesada, de forma que se continue el trabajo desde donde se dejó. En la figura 3.8 se muestra el esquema en el que se salvan a base de datos las alertas o eventos generados por Snort, OSSIM por ejemplo utiliza esta estructura.

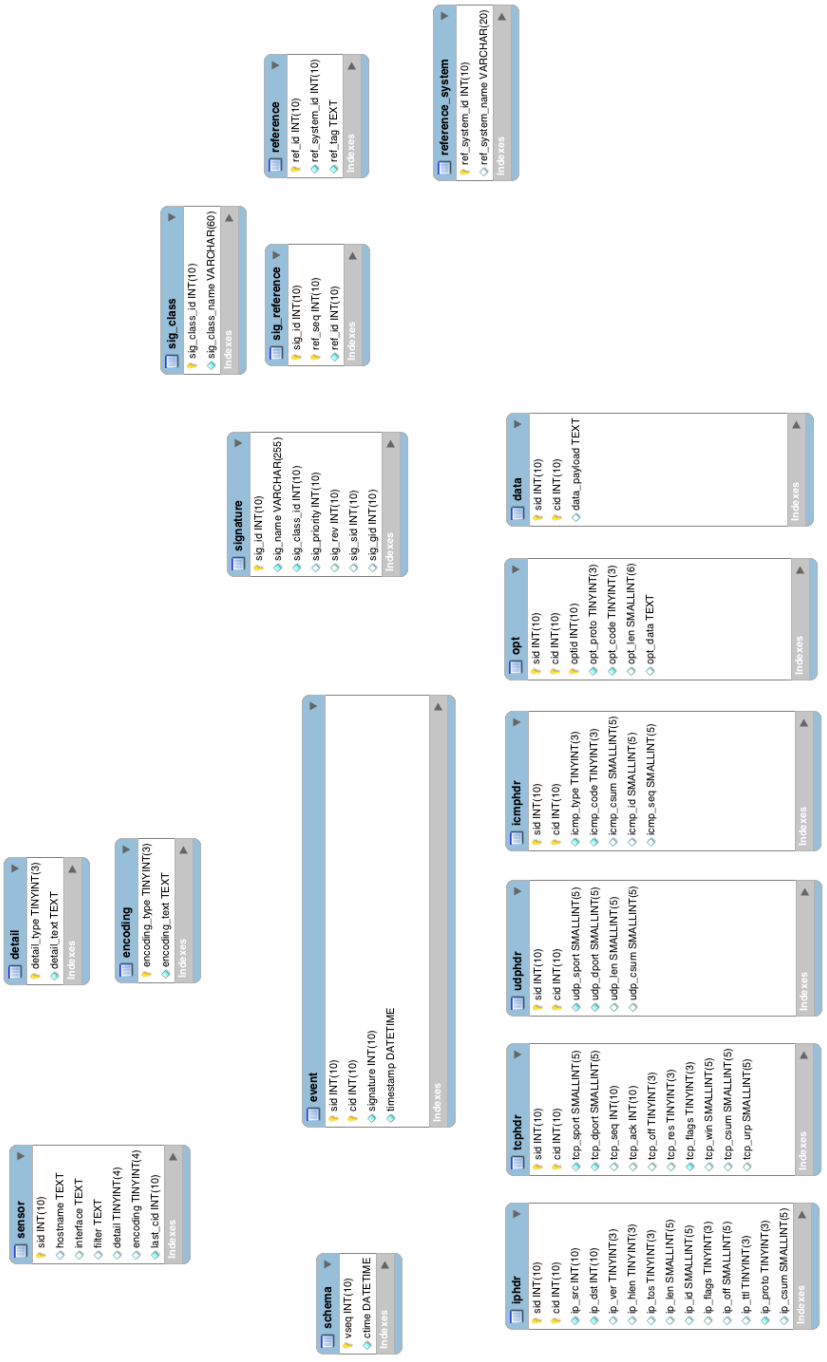


Figura 3.8: Esquema de Base de Datos relacional para Alertas/Log de Snort.

3.5 **Pentesting.**

En esta sección se va a hablar de las herramientas más comunes que se emplean para auditar los sistemas de Seguridad (*White-Hat*) en forma de lo que se denomina *pentesting*. Estas herramientas son usadas para recabar información de los analistas en Seguridad, viendo la posible situación en la que se encuentra la empresa auditada. Son herramientas comúnmente utilizadas también por los desarrolladores de *malware*, por consultores independientes que trabajan en la extracción de información de redes / productos empresariales (*Gray-Hat*) para la venta posterior, e incluso por grupos *hacker* (*Black-Hat*) con voluntad malintencionada.

Su finalidad principal es la recogida de información, si bien en base a la misma, se pueden lanzar ataques para explotar vulnerabilidades conocidas, o incluso generar los ataques por medio de *shellcodes* genéricas que tienen en su base de datos de conocimiento.

3.5.1 **Nessus.**

Nessus [tns15] es una herramienta de escaneo de vulnerabilidades principalmente. Su arquitectura se separa en dos partes, la primera una interfaz web avanzada que se comunica con la segunda, el escáner propiamente dicho. Cada tipo de auditoría que se realiza desde Nessus representa a un determinado *plugin*, teniendo más de 70.000 diferentes en su base de datos de conocimiento.

Nessus incluye las siguientes plantillas para realizar el escaneo de los recursos existentes:

- Escaneo avanzado: Se permite controlar completamente la configuración.
- Auditado de infraestructura cloud: Se usa para auditar la configuración de servicios en la nube proporcionados por terceros.
- Detección de *Bash Shellshock*.
- Escaneo básico de red.
- Logeo en los sistemas operativos para enumerar actualizaciones de software no realizado.
- Detección de *glibc GHOST*.

- Descubrimiento de hosts.
- Escaneo de dispositivos móviles.
- Auditado de ficheros de configuración de dispositivos de red.
- Auditado de cumplimiento de políticas proporcionadas por el usuario.
- Testeo de aplicaciones Web.
- Escaneo de Malware en sistemas Windows.

En las figuras 3.9 y 3.10 se puede ver un escaneo completo del entorno de laboratorio que se ha desplegado para la presente tesis. en especial es relevante ver la información para una distribución *metasploitable* (sección 3.5.4) que está instalada y funcionando entre las máquinas vulnerables.

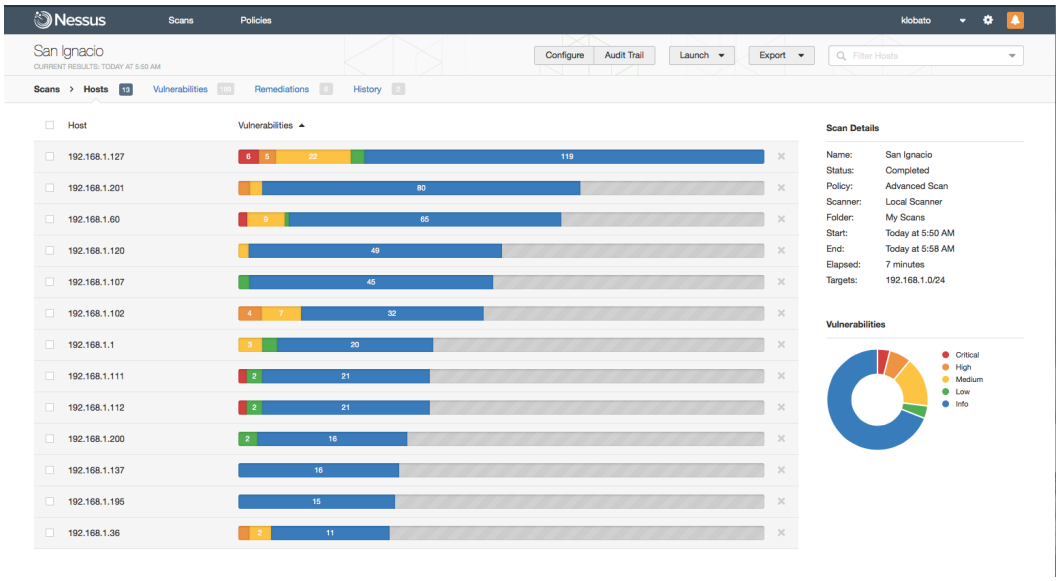


Figura 3.9: Nessus - Escaneo Laboratorio Tesis.

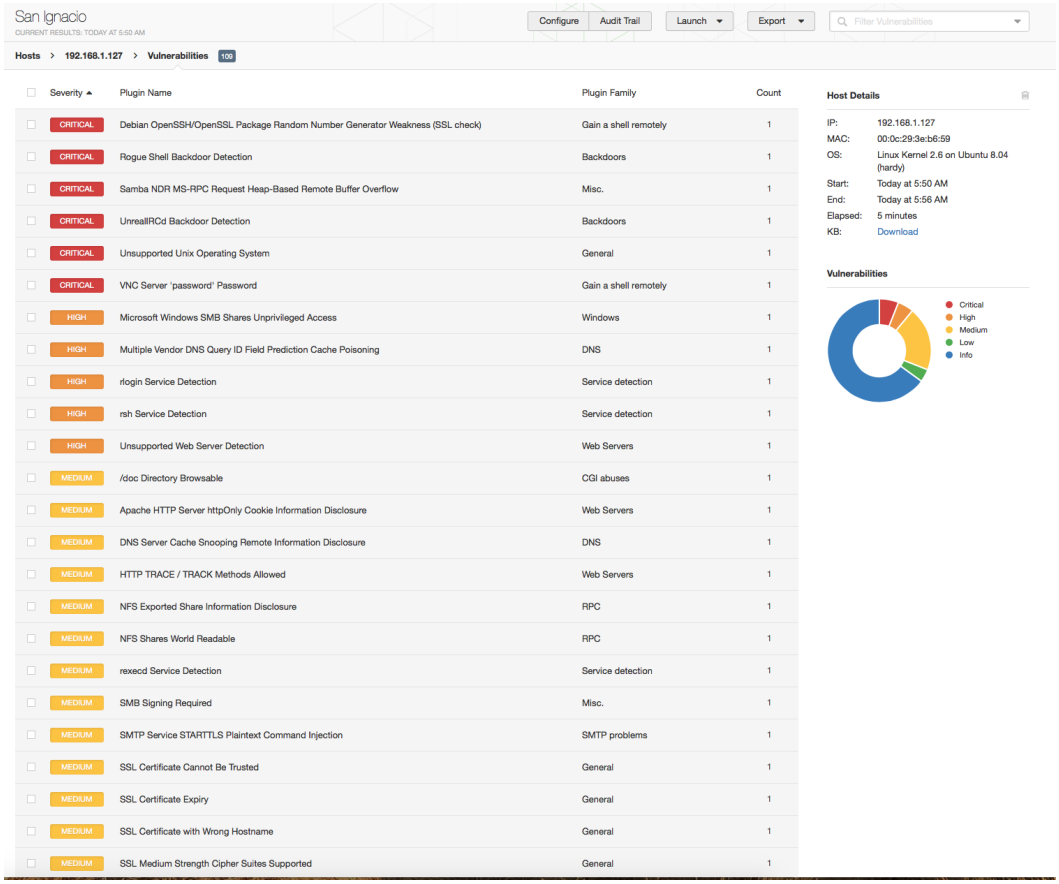


Figura 3.10: Nessus - Resultados para distribución Metasploitable.

3.5.2 **Mestasploit.**

La herramienta Metasploit [Rap15] es uno de los referentes dentro del área de auditorías de seguridad, proporcionando un espacio de trabajo intuitivo que se puede usar para tareas administrativas y para realizar pruebas de seguridad. Entre sus características más destacadas están:

- Escaneo de Hosts. Al igual que en la herramienta anterior, Nessus, se hace un barrido de los diferentes equipos que hay en la red, proporcionando información de los mismos y vulnerabilidades asociadas.
- Explotación. Metasploit dispone de módulos que proporcionan soporte para localizar las vulnerabilidades y para realizar una "explotación" de las mismas, teniendo soporte para módulos genéricos con *exploits*, módulos auxiliares, y módulos para después de la realizar la explotación.

En la figura 3.11 se puede ver el primer paso de escaneado de los hosts, información que una vez recabada será presentada de forma resumida al auditor. Una vez identificados los servicios accesibles, tipos de hosts, sistemas operativos, etc., el siguiente paso que permite la herramienta es la "explotación", o lo que es lo mismo: usar una batería de vulnerabilidades conocidas contra los servicios identificados, permitiendo en aquellos afectados positivamente la inyección de una *shellcode* de entre la enorme variedad que dispone *Metasploit Framework*. El rastreo de *exploits* se puede ver en la figura 3.12.

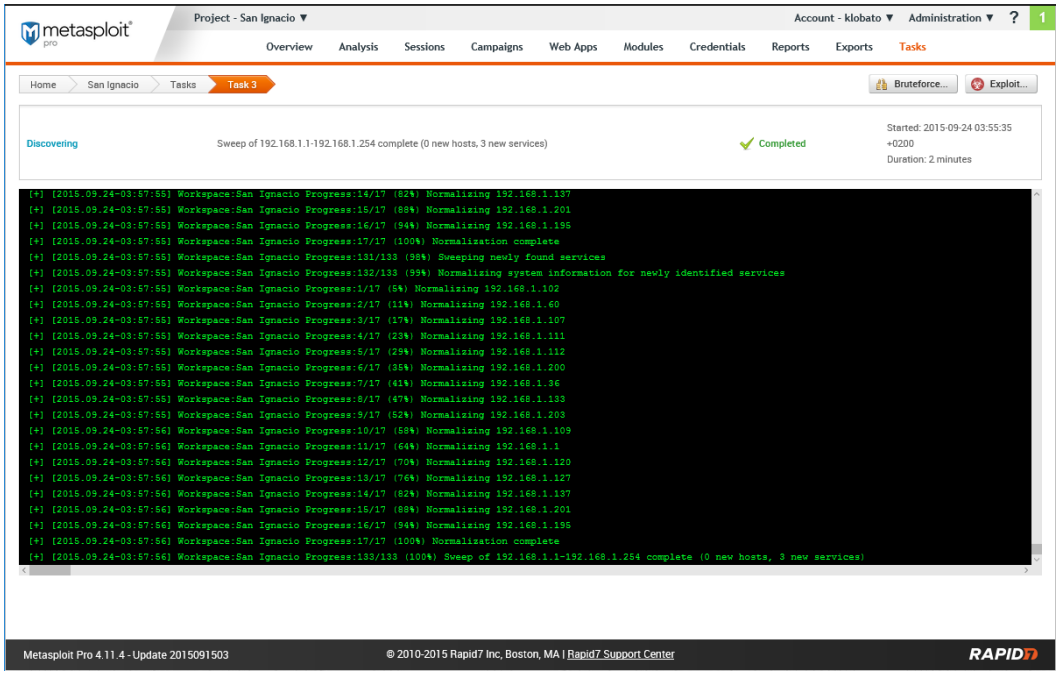


Figura 3.11: Metasploit - Descubrimiento de equipos en red.

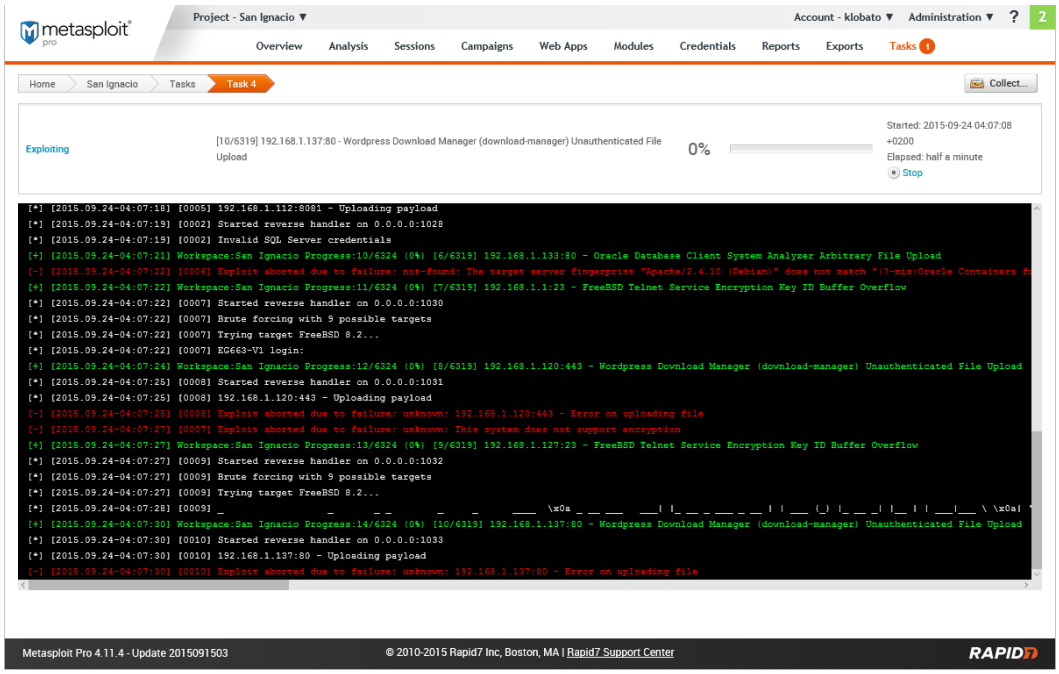


Figura 3.12: Metasploit - Búsqueda de vulnerabiildades que puedan ser explotadas.

En la figura 3.13 se muestra las diferentes acciones que permite realizar en el hosts haciendo uso de la combinación de *exploits* - *shellcodes*, que se pueden elegir de una lista bastante grande para el tipo de *exploit*, figura 3.14.

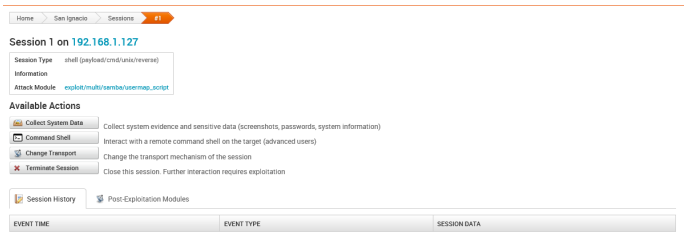


Figura 3.13: Metasploit - Acciones para un Host vulnerable.

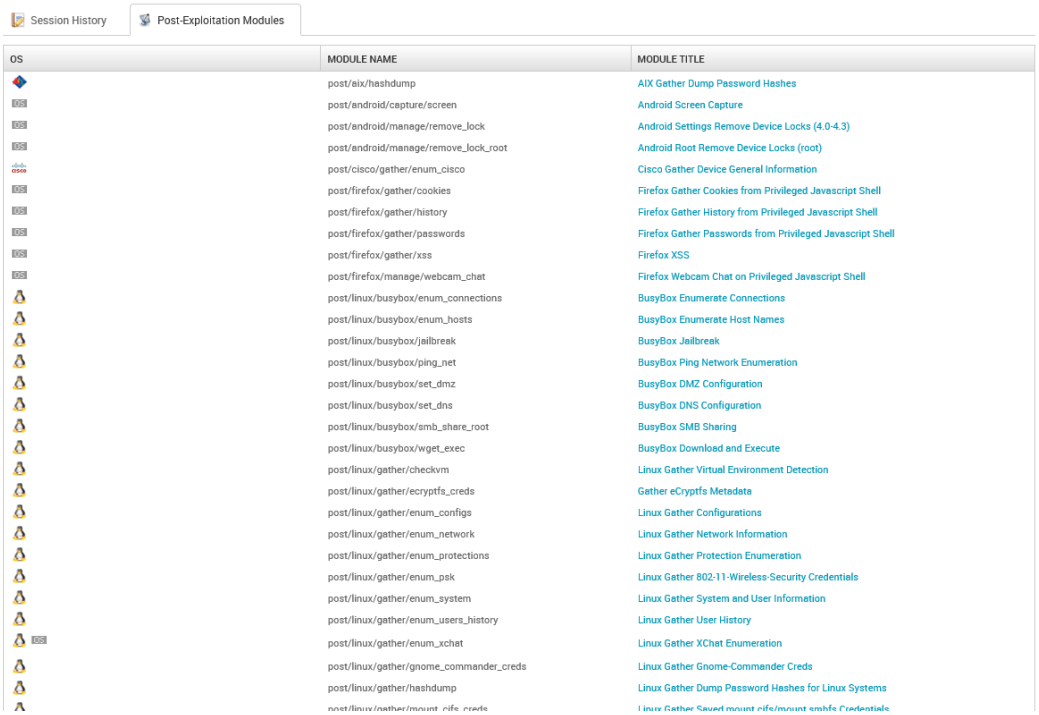


Figura 3.14: Metasploit - Módulos para post-exploitation.

Esta es una de las herramientas más usadas con diferencia y que reúne uno de los mayores conocimientos de ataques, vulnerabilidades, *exploits* conocidos, *shellcodes*, así como numerosas y diversas herramientas para la recogida de información de los equipos.

3.5.3 **Kali.**

La distribución *Kali Linux 2.0* [Sec15] (figura 3.15) es una distribución linux basada en *Debian* con el objetivo de facilitar las tareas de auditoría y de *pentesting*. Dispone en su interior de centenares de herramientas para cubrir varias fases de tareas relativas al área de seguridad: *Penetration Testing*, Análisis Forense (*Forensics*) e Ingeniería Inversa (*Reversing*). El origen de esta distribución viene de la antigua *BackTrack*, una distro que tenía objetivos similares hace años que salió de las conferencias dadas en *Black Hat* [Tec15].

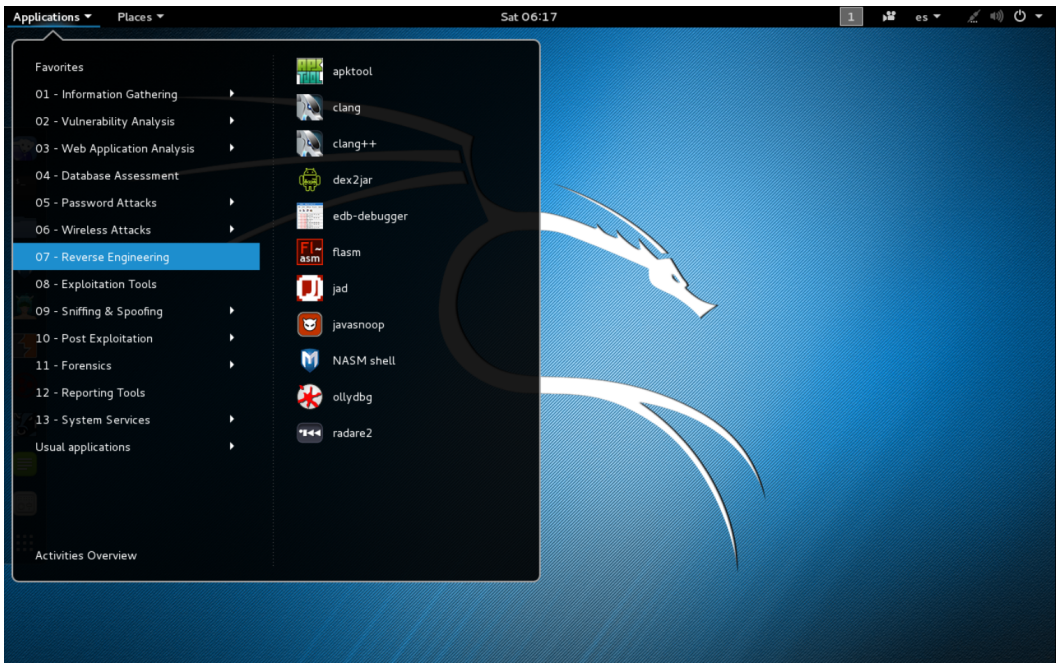


Figura 3.15: Distribución Kali Linux 2.0.

3.5.4 **Laboratorio para Pentesting.**

Para tener un entorno completo se requieren una serie de sistemas que sean vulnerables, sobre las que poder ejecutar las herramientas de *pentesting* anteriores y poder recabar la información correspondiente a los ataques. Con ello se generará el corpus necesario para el resto de los experimentos de la tesis, teniendo tráfico limpio y tráfico con ataques y pudiendo demostrar tanto la recogida de evidencias a sistemas *Big Data*, capítulo 6, como crear el *DataSet* que se usará en la capa de machine learning, capítulo 7.

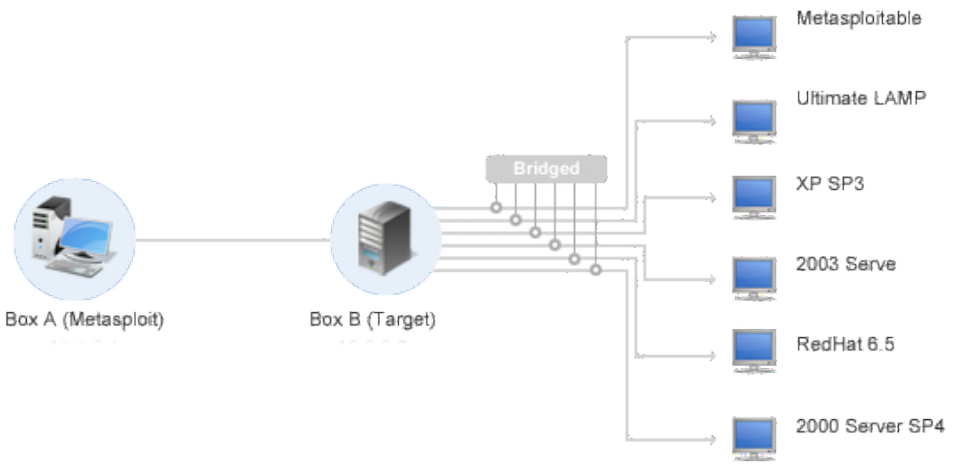


Figura 3.16: Ejemplo de laboratorio de *Pentesting*.

Hay varias distribuciones disponibles que pueden usarse para este fin, como se muestra en la figura 3.16. La imagen más usada para entrenamiento académico del área de Seguridad de la Información suele ser la denominada *Metasploitable 2*. Es una máquina virtual que se puede descargar de la propia página de Rapid7 - Mestasploit, que tiene numerosos bugs conocidos y que permiten la demostración de múltiples técnicas de ataque. Entre ellos:

- Servicios mal configurados que permite acceso y login remoto: *rlogin*, *rpcinfo*.
- Servicios con *backdoors* conocidas: VSFTPD, UnrealIRCD.
- Servicios con *backdoors* no intencionadas: distcc, samba, etc.
- Claves de las cuentas de usuario/sistema débiles.
- Servicios web vulnerables: phpMyAdmin, tikiwiki, Webdav, Damn Vulnerable Web Application, Multilidae.

Así mismo, otras distribuciones, bien en windows o bien en linux, que en el pasado tuvieron su presencia en las empresas, al quedarse desactualizadas contienen numerosos bugs y vulnerabilidades no solventadas. A tal efecto: Ultimate LAMP, RedHat 6.5, Windows en versiones obsoletas como Windows XP SP3, 2003 Server, 2000 Server SP4, etc. Al no instalar los últimos services packs, bien por descuido o bien por obsolescencia, todas las

versiones de Windows acaban siendo vulnerables, debido a que es un sistema operativo muy explotado por la comunidad *hacker*. Lo mismo ocurre con los servicios Web o las distribuciones Linux que no han sido actualizadas debido a bugs conocidos, generando un entorno excelente sobre el que realizar pruebas de laboratorio.

«Divide et vinces (Traducción: Divide y vencerás).»

Julio César (100a.C - 44a.C) -
Napoleón Bonaparte (1769 - 1821)

Capítulo

4

Arquitectura Propuesta ESIDE-BIDS.

La presente tesis basa su actividad principalmente en el área de *Big Data*, aplicada a la Seguridad de la Información, por lo que en este capítulo se realizará una descripción de las tecnologías existentes y su aplicación de cara al análisis de la información en el campo de estudio. Finalmente se detallarán las arquitecturas susceptibles de aplicación para los Sistemas Integrales de Seguridad de la Información, integrándose con los sistemas SIEM y/o otras tecnologías para adaptar flujos de datos provenientes de aplicaciones de seguridad.

4.1 El ecosistema Apache Hadoop

Hadoop, creado por Doug Cutting, comenzó con el proyecto *Nutch* (2002), que tras la publicación de *Google File System* en 2003 [GGL03] y *MapReduce* [Web04] en 2004, adoptaron estos nuevos paradigmas en el proyecto. Pasado el tiempo *Yahoo!* se interesó en el proyecto y financió la creación de un equipo de trabajo que acabarían generando el proyecto denominado Hadoop. Doug también creó el proyecto Apache Luccene que trata sobre minería de datos y algoritmos de *machine-learning*. En 2008 salió a la luz el proyecto Hadoop como un nuevo paradigma de procesamiento de gran-

des volúmenes de información, originalmente orientado a los motores de búsqueda *on-line* en la Web.

Hadoop supera a los sistemas relacionales de bases de datos, por medio de lo que se conoce como No-SQL, debido a que los tiempos de búsqueda que proporcionan los discos duros son menores que los tiempos de transferencia de los mismos. Si el patrón de acceso a los datos se basa en la búsqueda, se tardaría más en leer o escribir grandes porciones de datos que el pasar por todos los datos, ya que esto último se haría a la máxima tasa de transferencia. Así mismo, para actualizar una pequeña porción de los datos, la estructura tradicional de los sistemas es el algoritmo B-Tree y funciona mejor para búsquedas, si bien cuando se tienen que realizar actualizaciones masivas, la estructura B-Tree es menos eficiente que el algoritmo MapReduce, que hace uso de *Sort/Merge* para reconstruir la base de datos.

En la mayoría de los casos, los sistemas basados en MapReduce son complementarios de los sistemas de bases de datos relacionales (RDBMS). La comparación se puede encontrar en la siguiente tabla 4.1.

Cuadro 4.1: Comparación de MapReduce con Bases de Datos Tradicionales.

	Bases de Datos Tradicionales	**MapReduce**
Tamaño Datos	GigaBytes	PetaBytes
Acceso	Interactiva y en lotes	Proceso en lotes
Actualización	Múltiples lecturas y escrituras	Una escritura, múltiples lecturas
Transacciones	ACID	Ninguna
Estructura de Esquema	En la escritura	En la lectura
Integridad	Alta	Baja
Escalabilidad	No-Lineal	Lineal

En resumen MapReduce es una técnica muy buena para aplicaciones en las que los datos se escriben una vez y se leen muchas, mientras que una base de datos es perfecta para escenarios en los que los *datasets* se actualizan continuamente. Si bien, las limitaciones de ambos sistemas han ido evolucionando, haciendo que las bases de datos tradicionales integren ideas de Hadoop, y que Hadoop integre sistemas como Hive para añadir características como índices y transacciones.

Otra diferencia entre Hadoop y los sistemas de bases de datos tradicionales radica en el tamaño que necesitan para mantener la estructura de los

datos sobre los que operan. Los datos denominados como **estructurados** se organizan a través de un esquema predefinido. Los datos conocidos como **semiestructurados** tienen una estructura, si bien generalmente se obvia, un ejemplo: una hoja de datos en la que se organiza la información en celdas, si bien éstas pueden contener cualquier tipo de información. Los datos **no estructurados**, en cambio, no tienen ningún tipo de estructura interna, por ejemplo: imágenes o texto plano. Los datos relacionales generalmente requieren de un proceso denominado **normalización** para mantener la integridad y eliminar la redundancia. MapReduce en cambio escala linealmente en relación al tamaño de los datos, éstos son particionados dentro del clúster y en base a las primitivas de Map y Reduce, se trabaja con ellos por lotes de forma paralela.

El paradigma de MapReduce funciona haciendo que el análisis de la información sea en lotes, es decir, se distribuye un *DataSet* en los nodos que integran el clúster, haciendo que cada nodo tenga una porción del mismo. Mediante un controlador se hace que cada nodo analice localmente la información que tiene en su sistema de archivos, *"map"*, enviando los resultados al controlador central, quien se ocupará de combinar los datos que llegan, *reduce"*, para dar la salida final a los mismos. Un flujo básico de M-R [1] se puede ver en la figura 4.1.

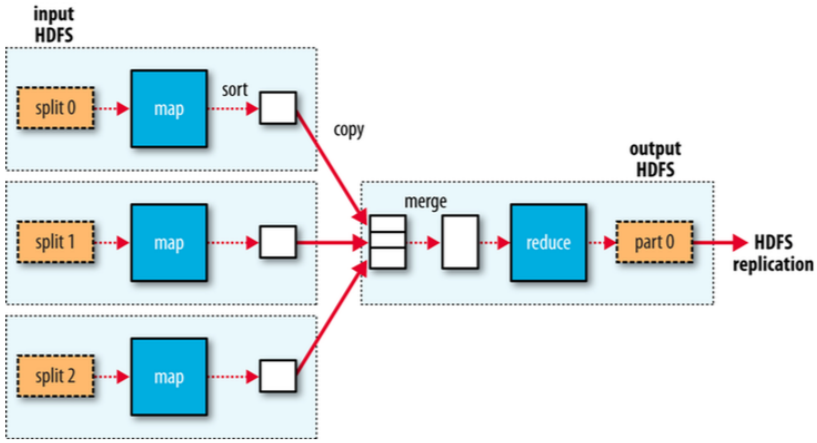


Figura 4.1: Ejemplo del flujo de datos en M-R.

[1] M-R: MapReduce.

4.1.1 HDFS. El sistema de archivos distribuido de Hadoop.

El sistema de archivos HDFS es quizá el punto más importante dentro del proyecto Hadoop, ya que es quien se ocupa de almacenar grandes volúmenes de información organizada en ficheros usando para ello *Comodity Hardware*[1] [SKRC10]. HDFS trata de dar soporte a los siguientes requisitos:

- Almacenamiento de ficheros grandes: Los ficheros que se van a almacenar puede tener desde unos pocos megabytes a gigabytes, e incluso terabytes de tamaño. Actualmente hay clústers de Hadoop funcionando que tienen PetaBytes de datos en su interior.

- Acceso a los datos en flujos: HDFS se fundamenta en la idea de que el patrón de explotación de datos más eficiente es el de escribir una vez y leer múltiples veces. Así pues un *dataset* se genera o se copia una vez desde la fuente y después se practican varios análisis sobre el mismo a lo largo del tiempo. En general, se usa el *dataset* al completo, por lo que es más significativo el tiempo en leerlo al completo que el tiempo en acceder al primer registro.

Así mismo, también conviene tener en cuenta las limitaciones que tiene HDFS, para poder determinar las aplicaciones que debería evitar este almacenamiento:

- Acceso a datos de baja latencia: Las aplicaciones que requieren de una baja latencia en el acceso a los datos, del orden unos pocos milisegundos, no deberían usar HDFS. Para estas aplicaciones HBASE (sección 4.8) es una opción mucho más recomendable.

- Gran cantidad de ficheros pequeños: Debido a que el nodo central mantiene los metadatos del sistema de archivos HDFS en memoria, el límite del total de archivos que puede tener el clúster en HDFS viene fijado por la cantidad de memoria que tenga el controlador principal ó *namenode*. A modo de regla de pulgar: cada fichero, directorio o bloque ocupa unos 150 bytes.

- Muchas fuentes escribiendo, haciendo modificaciones arbitrarias: Los ficheros en HDFS suelen ser escritos por un único escritor. Las escrituras se hacen al final del archivo, a modo de *append*, así que no hay

[1] Se entiende por *Commodity Hardware* todo aquel hardware de servidores o PCs que es de uso general, sin costes elevados como puedan ser los Servidores Blade, los Supercomputadores avanzados o las cabinas de discos basadas en fibra óptica (HP EVAs, etc.).

soporte para múltiples escritores en el mismo descriptor de fichero, y menos aún para modificaciones en offset diferentes. Este tipo de funcionalidades harían al sistema muy ineficiente.

4.1.1.1 **Conceptos fundamentales de HDFS.**

4.1.1.1.1 Bloques

En un disco, se entiende como bloque la mínima cantidad de datos que puede escribir o leer. Los sistemas de ficheros se basan en este concepto de cara a organizar los datos, el sistema de archivos al completo es generando mediante la integración de múltiples bloques. Los bloques de los sistemas de ficheros suelen ser de alrededor de unos pocos kilobytes, mientras que el bloque de un disco suele ser de 512 bytes. HDFS fija el tamaño de bloque en 128 MB por defecto, y como en un sistema de ficheros tradicional los datos se rompen en porciones del tamaño del bloque de ficheros. La principal diferencia de HDFS es que si un fichero ocupa menos que el bloque fijado, no dejará el resto del bloque vacío, si no que ocupará lo correspondiente al archivo que almacena.

En HDFS el tamaño de bloque es tan grande para minimizar el coste de las búsquedas. Si el bloque es lo suficientemente grande, el tiempo que lleva el transferir la información desde el disco es mayor que el tiempo que cuesta encontrar el principio del bloque. Por ello, basándose en la premisa de que el tiempo de búsqueda tiene que ser el menor de todos, más grande que el tiempo de transferencia, para poder asegurar que el fichero se evalúa a la tasa máxima de transferencia, los tiempos de búsqueda tienen que ser lo menores posible.

Así mismo, al ser un sistema de archivos distribuido y tener los datos repartidos en múltiples equipos en la red, implica que un fichero repartido en bloques de HDFS puede ser más grande que cualquier disco unitario de los nodos que lo almacenan. La simplificación que ofrece el tener los ficheros repartidos en bloques es debida a que se elimina el concepto de *metadatos*, ya que los bloques son sólo porciones de datos a almacenar, y se delega en otros sistemas para mantener esta información. Finalmente otro de los puntos críticos de los bloques en HDFS es el hecho de que proporcionan tolerancia a fallos y alta disponibilidad. Cada uno de los bloques generalmente es replicado a un número mayor de uno de nodos en el clúster, haciendo que si los datos se corrompen se pueda restaurarlos, o caso de que algún nodo del clúster falle, siempre habrá otros que puedan ofrecer estos bloques a las aplicaciones de procesamiento.

4.1.1.1.2 Namenodes y Datanodes

Como se puede intuir por los comentarios realizados en el punto anterior, el sistema de archivos se fundamenta en el patrón de maestro-esclavo, donde un maestro o *namenode* controla a un número de esclavos, *datanode*. El *namenode* se ocupa de la gestión del sistema de ficheros, manteniendo el árbol de archivos y los metadatos asociados. Esta información, que como se ha comentado en funcionamiento se mantiene en memoria, se persiste a disco también en forma de dos ficheros: la imagen del *namespace* y el fichero de transacciones. El nodo central dispone de la información de todos los nodos secundarios, *datanodes*, y los bloques de fichero que almacenan, y esta información se recupera en el arranque de cada uno de ellos. Todo el sistema de archivos sigue las reglas de forma similar a POSIX, de forma que el usuario no tenga que conocer nada de esta estructura de información para trabajar con los ficheros.

Lógicamente si el *namenode* falla y se corrompe, no hay forma de recuperar la información que se encuentra dentro del sistema de archivos, así que por ello, Hadoop implementa varios mecanismos para evitar y subsanar este tipo de situaciones:

- Almacenar los ficheros persistentes de la información del *namenode* en otro emplazamiento, como por ejemplo una unidad de red remota NFS.

- Poner en funcionamiento un segundo *namenode* que si bien no actúa como controlador, periódicamente recupera la información de la imagen del sistema de archivos para prevenir que el fichero de transacciones sea demasiado grande o viejo. Este segundo controlador generalmente se despliega en otra máquina del clúster, aunque en caso de fallo del controlador principal la pérdida de datos está garantizada, por lo que el procedimiento habitual es copiar los datos que se tienen en la unidad remota de red del punto anterior y hacer que el segundo controlador arranque como primario.

- Se puede ejecutar otro *namenode* en formato de alta disponibilidad, que se ampliará en la sección 4.1.1.1.5.

4.1.1.1.3 Cacheo de bloques.

Generalmente un *datanode* lee los bloques del disco directamente, pero aquellos que son accedidos frecuentemente pueden ser explícitamente ca-

cheados en la memoria del host. Por defecto, un bloque sólo se cachea en un único *datanode*, pero este número es un parámetro configurable.

4.1.1.1.4 HDFS Federation

Como se ha indicado, el *namenode* se ocupa de mantener la referencia a todos los bloques que hay en el sistema de archivos, y toda esta información se almacena en memoria, por lo que este último recurso limita el tamaño máximo del sistema de archivos que se puede crear en un clúster. La característica *HDFS Federation* se introdujo a partir de la versión 2.x de Hadoop, y permite escalar un clúster por medio de la adición de *namenodes* al mismo. Los *namespaces* o las rutas de montaje dentro del sistema de archivos son independientes unas de otras, pero para los usuarios finales es un sistema de archivos distribuido único.

4.1.1.1.5 HDFS y la Alta Disponibilidad.

Se ha visto anteriormente que el *namenode* es un punto de fallo crítico. Hadoop 2.x soluciona esta limitación por medio de añadir alta disponibilidad, haciendo que existan dos *namenodes* en una configuración de activo/a la espera. Los cambios que se introducen son los siguientes:

- Los namenodes utilizan un almacenamiento distribuido para compartir los datos persistentes.
- Los datanodes deben enviar reportes de los bloques que contienen a ambos namenodes.
- Los clientes se deben configurar para lidiar con posibles fallos en el namenode, que es un mecanismo en general transparente para los usuarios.
- El rol del namenode secundario es traspasado a este nuevo tipo de namenodes de alta disponibilidad, que también hacen copias periódicas de la actividad del namenode principal.

Para compartir la información de alta disponibilidad entre los namenodes, hay varias alternativas, siendo una de ellas los sistemas remotos de ficheros como NFS, o lo que se denomina QJM (*Quorum Journal Manager*). La opción que se implementa habitualmente en los clústers es la segunda y básicamente consiste en lo siguiente:

- Se almacena en Zookeeper (sección 4.7) la información sobre los controladores de *failover*, y cada namenode se ocupa de controlar a sus otros namenodes para monitorizar que no haya fallos en los mismos.
- QJM sólo permite que haya un namenode escribiendo en los datos persistentes del sistema de ficheros (metadatos).
- El failover se realiza de forma transparente a las librerías de cliente que se proporcionan para acceder a los ficheros y la propia URL del sistema de archivos HDFS puede llevar asociada las direcciones de los namenodes configurados en alta disponibilidad.

4.1.2 YARN. Gestor de recursos de Hadoop.

A partir de la versión 2.x de Hadoop se introdujo un nuevo gestor de recursos, que proporciona APIs para solicitar y trabajar con los recursos que tiene el clúster, si bien estas APIs no son usadas directamente por el usuario final. Los usuarios escriben aplicaciones en APIs de más alto nivel que proporcionan los frameworks de computación distribuida, y son estos últimos quienes hacen uso de la capa YARN (*Yet Another Resource Negotiator*):

- Aplicaciones high-level: Pig, Hive, Crunch, etc.
- Aplicación low-level: MapReduce, Spark, Tez, etc.
- Procesamiento: YARN.
- Almacenamiento: HDFS, HBASE.

YARN se ocupa de proporcionar sus servicios por medio de *daemons* que están permanentemente corriendo en los nodos del clúster: un gestor de recursos (*resource manager*) para todo el clúster que se ocupa de la gestión de estos, y los gestores de nodo (*node managers*) que lanzan y monitorizan los "contenedores". Un contenedor no es más que una aplicación con un conjunto de recursos asignado (memoria, CPU, etc.).

Para lanzar una aplicación en YARN, el cliente contacta con el gestor de recursos y le pide el lanzamiento de una aplicación. Es el gestor de recursos quien encuentra los gestores de nodo, quienes serán finalmente los que lancen la aplicación con los recursos que se hayan asignado para la misma.

Un dato importante a tener en cuenta es que YARN no proporciona mecanismos de comunicación entre las partes de una aplicación: cliente, maestro y proceso; haciendo que para esto último se hagan uso de otros tipos de

comunicación como Hadoop RPC para enviar resultados o actualizaciones entre sí.

Para la gestión de recursos y la planificación de los mismos, YARN ofrece tres posibilidades:

- **El planificador FIFO**. Es el más sencillo de entender y no requiere de ninguna configuración adicional. Pone las aplicaciones en una cola y las pone a funcionar en el orden en el que han ido llegando a la misma. Si bien este planificador no suele ser el más conveniente para clústers grandes compartidos.

- **Planificador basado en la capacidad**. Con este planificador un trabajo pequeño puede empezar tan pronto se crea, ya que se almacena en una cola separada. Si bien este planificador implica que las aplicaciones más pesadas serán las que más tarde van a lanzarse y finalizar.

- **Planificador justo**. Asigna los recursos del clúster en función de las aplicaciones que tenga en la cola. Si se lanza una aplicación grande usará todos los recursos mientras no haya más aplicaciones en funcionamiento, pasando a ocupar menos recursos cuando llegan otras aplicaciones más pequeñas.

4.1.3 Apache Ozzie. Gestor de Flujos.

Es un sistema para lanzar flujos de trabajo, aplicaciones, de trabajos dependientes. Se compone de dos partes principales: el motor de flujos, que almacena y pone en funcionamiento los flujos compuestos de trabajos de Hadoop (MapReduce, Pig, Hive, etc.); y el motor de coordinación que se ocupa de lanzar los flujos en base a las planificaciones establecidas y la disponibilidad de los datos.

Ozzie ha sido diseñado para escalar y puede manejar la ejecución temporal de miles de flujos en un clúster de Hadoop, cada uno de ellos compuesto por decenas de trabajos. Así mismo, simplifica la trazabilidad del lanzamiento de tareas ya que no requiere que se vuelvan a lanzar aquellas tareas que han sido ejecutadas correctamente.

Ozzie corre como un servicio dentro del clúster y sus clientes pueden proporcionar flujos que se vayan a lanzar inmediatamente o más tarde acorde a una planificación. Un flujo en Ozzie se define como un Grafo Acíclico Dirigido de nodos de acción y nodos de control de flujo. Un nodo de acción representa una tarea dentro del flujo, como por ejemplo: mover ficheros a

HDFS, ejecutar una tarea de MapReduce, lanzar un proceso de Spark Streaming, una tarea de Pig o un trabajo de Hive. Un nodo de control se ocupa de gestionar la ejecución entre las acciones a través de una lógica condicional o ejecución paralela. Ozzie notifica a la finalización de una tarea de los resultados de la misma, y también es posible recibir actualizaciones cuando el flujo llega a un determinado paso o sale de un nodo de acción.

Las definiciones de flujo se realizan en lenguaje XML, usando el lenguaje denominado *Hadoop Process Definition Language*, cuya especificación se puede encontrar en la página web de Ozzie.

4.1.4 **Apache AVRO. Lenguaje neutral para serialización de datos.**

El tener la información organizada en un formato de datos apropiado y que pueda ser procesado por muchos lenguajes: C, C++, C#, Java, JavaScript, Perl, PHP, Python, Ruby, etc.; facilita enormemente compartir *datasets* entre múltiples actores. Se pasa en otros sistemas de serialización como Apache Thrift o Google's Protocol Buffers, y al igual que en ellos los datos se describen en esquema independiente del lenguaje, pero a diferencia de otros sistemas la generación de código es opcional, es decir, que se puede leer y escribir datos que cumplen un esquema aunque el código no haya visto ese esquema con anterioridad. Los esquemas de AVRO generalmente se escriben en JSON y los datos son codificados utilizando un formato binario generalmente, aunque hay otros mecanismos.

Se dispone de un lenguaje de alto nivel, *Avro IDL*, para escribir esquemas en un lenguaje muy parecido a C. La especificación de AVRO define específicamente el formato binario y el resto de características que todas las implementaciones deberán soportar.

AVRO tiene unas capacidades a nivel de esquema muy altas, haciendo que el esquema usado para leer los datos pueda no ser idéntico al esquema usado en la escritura de los mismos. De esta forma, AVRO permite la evolución del esquema a lo largo de sus diferentes versiones.

Otra característica fundamental del proyecto Apache AVRO es la especificación del formato de los objetos contenedores, de forma que se soporta tanto la compresión, como la separación en fragmentos, lo que es fundamental para datos de entrada en procesos MapReduce, Spark, Pig, Hive, etc.

4.2 Apache Flume

Hadoop se diseño para el procesamiento de grandes volúmenes de datos y pese a que muchas veces se asume que esta información ya está presente en HDFS o puede ser copiada a este sistema de archivos, en muchos sistemas no se puede contar con ello. Hay sistemas que producen gran cantidad de información que puede requerir un tratamiento de la misma vía agregaciones, almacenamiento y análisis utilizando Hadoop, y es para este tipo de sistemas donde Apache Flume [Fou15e] es ideal.

Flume ha sido diseñado para la ingesta de gran volumen de información y la transferencia del mismo a Hadoop, generalmente orientado a sistemas que producen eventos. Un ejemplo simple: la recogida de los logs de una granja de servidores web. Flume puede realizar la recolección incremental de esos logs y los puede escribir en HDFS, o incluso en HBASE o Solr.

Para hacer funcionar a Flume se necesita de un agente, que no es más que un proceso Java que corre tanto en *sources* como en *sinks* y que se conectan por medio de "canales". Una fuente produce los eventos y los transmite al canal, almacenándose en el mismo hasta que sean propagados a "sumidero" o consumidor. Una instalación de Flume se basa en una colección de agentes conectados funcionando en una arquitectura distribuida topológicamente. Los agentes en el extremo del sistema recogen la información y la transmiten a los agentes responsables de agregar y almacenar la misma.

Flume dispone de transacciones separadas para garantizar la entrega desde la fuente al canal y desde el canal al consumidor. El canal puede configurarse como de tipo *file* o como de tipo *memory*, siendo la principal diferencia en que con el primer tipo no se perderá, mientras que con el segundo sí. Obviamente, el beneficio de usar el segundo tipo es debido al rendimiento que se puede obtener, muy superior al primero.

Flume intenta agrupar los eventos en lotes, de forma que para cada transacción, siempre que sea posible, se procese más de un fichero. El tamaño de lote se fija para cada aplicación y suele venir dado por el componente a transferir.

Un concepto importante es el hecho del **Particionamiento** (*Partitioning*), ya que por norma, con grandes volúmenes de información, ésta última se suele organizar en particiones, de forma que las consultas se puedan hacer sólo sobre una partición siempre que sean sobre el subconjunto de datos que mantiene. Así mismo, es una práctica común el realizar este particionamiento en función del tiempo.

El uso habitual de este sistema, si bien no está orientado al tiempo real,

suele ser para la recogida de ficheros de logs incrementales de otros equipos remotos. Un ejemplo de esto se puede ver en la figura 4.2.

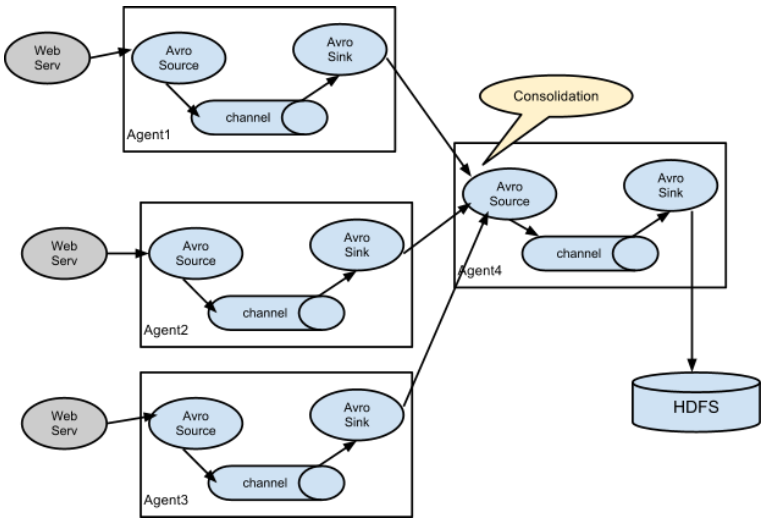


Figura 4.2: Ejemplo de arquitectura Apache Flume para recogida de WebLogs.

4.3 Apache Sqoop

Una de las mayores ventajas de Hadoop es poder trabajar con múltiples fuentes de datos, pero siempre es necesaria la interacción con otros repositorios que no sean HDFS. Los programas que corren en Hadoop requieren de APIs para poder acceder al exterior, habitualmente bien para traer información de sistemas RDBMS, bien para enviar información final procesada a los anteriores.

Apache Sqoop [Fou15i] es una herramienta *open-source* que permite a los usuarios extraer información de una base de datos estructurada en el entorno de Hadoop para posteriores procesamientos. Este procesamiento bien puede ser con MapReduce o con herramientas de más alto nivel como Hive. Incluso es completamente posible realizar la importación directamente contra HBASE, que se explicará en la sección 4.8. Los resultados finales de un *pipeline*, o flujo de trabajo, dentro de Hadoop pueden ser exportados también con Sqoop de forma que puedan ser explotados o consumidos por otros clientes tradicionales.

Actualmente se está trabajando en la versión 2 de la herramienta, que proporciona una API en Java que puede ser usada directamente por las aplicaciones, una interface Web, un cliente de consola (CLI, *Comand-line*

Interface) y una API REST. Esta segunda versión es la que se usa desde proyectos como Apache Spark (sección 4.10).

Sqoop funciona en base a "conectores", que cubren desde *Data Warehouses* corporativos como Teradata, bases de datos tradicionales (MS-SQL, Oracle, MySQL, PostgreSQL, etc.), e incluso almacenamiento No-SQL como Couchbase.

La importación que realiza Sqoop puede venir en varios formatos. El más común es el formato "texto", pero ya que este formato no puede contener campos binarios (columnas de base de datos tipo VARBINARY), o distinguir entre valores null o Strings que contengan el valor "null", se dispone de otros formatos más avanzados. Sqoop soporta: SequenceFiles, ficheros Avro y Parquet. Estos formatos binarios proporcionan la representación más precisa posible de los datos a ser importados; incluso soportando la compresión y la paralelización de los mismos para dar soporte a las aplicaciones de Hadoop.

Las figuras 4.3 y 4.4 muestran a modo de ejemplo el proceso de importación y exportación que permite la herramienta Sqoop.

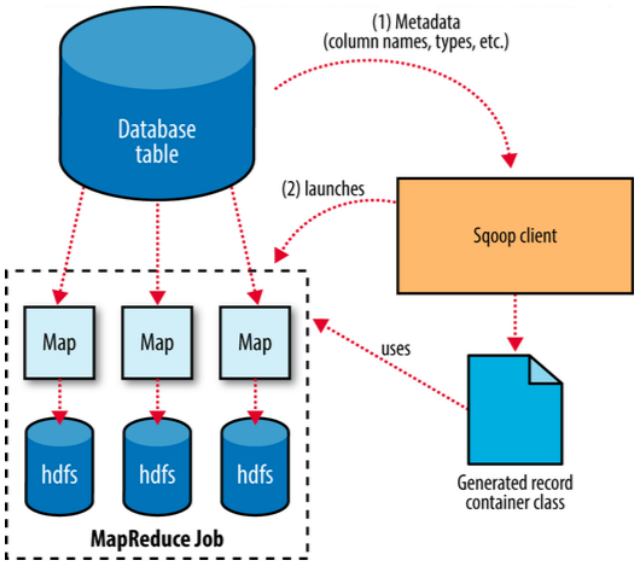


Figura 4.3: Sqoop: Ejemplo de importación de datos.

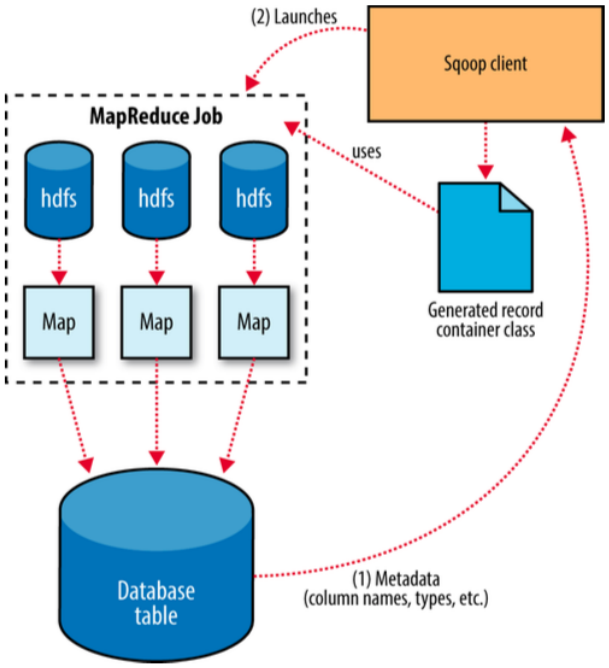


Figura 4.4: Sqoop: Ejemplo de exportación de datos.

4.4 Apache Pig

El proyecto Apache Pig [Fou15n] es un nivel de abstracción para el procesamiento de grandes volúmenes de información. Mientras que usando Hadoop de forma nativa, un programador puede definir una aplicación MapReduce, especificando la función que se va a aplicar en el *map*, así como la correspondiente a aplicar en el *reduce*, y combinar las tareas en múltiples etapas, esta actividad puede ser un reto para la gente de a pie. Pig viene a proporcionar unas estructuras de datos enriquecidas, generalmente siendo de múltiples valores y con niveles de anidamiento, permitiendo realizar transformaciones a los datos más potentes. Entre estas operaciones se incluyen *joins*, que como se ha comentado no son parte de la API MapReduce.

Pig se compone de dos partes bien diferenciadas:

- El lenguaje que se usa para definir los flujos de datos, denominado *Pig Latin* (ejemplo en listado de código 4.1).
- El entorno de ejecución que corre los programas en *Pig Latin*. Actualmente son dos entornos separados, uno para ejecución local y otro que corre en el entorno distribuido de Hadoop.

Un programa en *Pig Latin* consiste en una serie de operaciones, o transformaciones, que se aplican a los datos de entrada para producir una salida. Las transformaciones que se indican en el lenguaje *Pig Latin* se transforman por el framework en trabajos MapReduce, pero quedan totalmente abstraídos para los usuarios de Pig.

Pig no es más que un lenguaje de *scripting* que permite explorar grandes volúmenes de información. Mientras que el ciclo de desarrollo de MapReduce considerablemente largo: programar los *mappers* y los *reducers*, compilar y empaquetar el código, lanzar el trabajo y recoger los resultados; pudiendo ser un cuello de botella en muchas organizaciones, Pig tiene la habilidad de procesar terabytes de información en respuesta a unas pocas líneas de código enviado desde la consola. Pig fue creado en *Yahoo!* para facilitar las tareas de los investigadores y de los ingenieros, mejorando la velocidad con la que se puede analizar la información.

Pig permite ver las estructuras de los datos mientras se está escribiendo un programa, así como ver los resultados del mismo sobre un subconjunto de datos de entrada, pudiendo ver si hay errores en el procesamiento de la información antes de lanzar el proceso a todo el corpus del conjunto de datos completo (*dataset*).

En contrapartida, Pig si bien se acerca al rendimiento de los programas escritos directamente en MapReduce, no es tan eficiente como estos últimos, con lo que dependiendo del tipo de actividad a realizar puede no ser la mejor opción, si bien el tiempo que requiere lanzar consultas complejas usando Pig no es comparable con el desarrollo de aplicaciones a medida para lograr el mismo análisis.

Hay tres formas distintas de ejecutar los programas Pig:

- ***Script***: Pig puede correr un fichero de script que contenga los comandos.
- ***Grunt***: Esta es la *shell* interactiva para ejecutar comandos Pig.
- ***Embedded***: De la misma forma que se usa JDBC en Java para lanzar programas SQL, se puede usar *PigServer class* desde Java para lanzar aplicaciones Pig.

Un ejemplo de un programa en Pig:

Código 4.1: Ejemplo de Pig Latin

```
records = LOAD 'input/blabla.txt'
        AS (year:chararray, temperature:int, quality:int);
```

```
filteredRecords  = FILTER records BY temperature != 9999 AND quality IN
    (0,1,4,5,9);
grouped_records = GROUP filtered_records BY year;
max_temp = FOREACH grouped_records GENERATE group,
    MAX(filtered_records.temperature);
DUMP max_temp;
```

El lenguaje de Pig es similar al lenguaje SQL, si bien la principal diferencia es que el primero es un lenguaje de definición de flujos, mientras que el segundo es un lenguaje declarativo.

4.5 Apache Hive

Este proyecto [Fou15g] fue creado por Jeff Hammerbacher y su equipo de desarrolladores en FaceBook para ser desplegado en lo alto de Hadoop como el framework de referencia para el *Data Warehousing*. Hive se diseñó para los analistas con altas capacidades en SQL para ejecutar consultas sobre volúmenes de datos elevados. Si bien SQL no es ideal para todos los problemas de Big Data, por ejemplo sería muy complicado implementar algoritmos de *machine-learning*, sí lo es para todos aquellos proyectos que están orientados a herramientas de *Business Intelligence*.

Los clientes de Hive son los siguientes:

- ***Thrift Client***. El servidor de Hive se expone como un cliente *Thrift*, de forma que es posible interaccionar con él usando cualquier lenguaje de programación que lo soporte.
- ***JDBC Driver***. Hive proporciona un cliente tipo 4 del driver JDBC (completamente Java). Una aplicación puede conectarse a la uri del servicio Hive, pudiendo hacer uso de las funcionalidades de JDBC. El cliente *BeeLine* CLI que proporciona Hive para realizar consultas en la consola funciona internamente con este sistema.
- ***ODBC Driver***. Un driver ODBC permite a las aplicaciones comunicarse a través del protocolo ODBC, permitiendo al software de *Business Intelligence* conectarse a Hive. Esta opción está disponible en base a terceros vendedores.

En la figura 4.5 se puede ver la arquitectura de Hive, incluyendo los clientes y cómo acceder al resto de la infraestructura Hive (generalmente a través de *Thrift*).

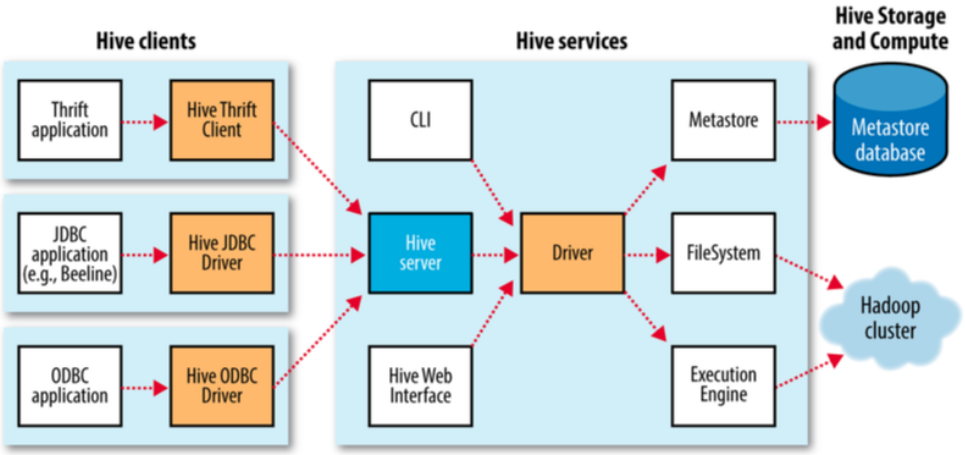


Figura 4.5: Arquitectura de Hive.

4.5.1 El Metastore

Es el repositorio principal de los metadatos de Hive. Se compone de dos partes: un servicio y el almacenamiento de los datos. Generalmente el servicio corre en la misma máquina en que está corriendo el servicio de Hive y contiene una base de datos embebida Derby, salvaguardada en disco. Una limitación es que sólo se puede tener una sesión de Hive abierta accediendo al mismo *Metastore*.

La solución para poder tener varias sesiones es usar una base de datos tradicional, haciendo que el servicio de metastore, pese a correr en la misma máquina que el servicio de Hive, se conecte a una base de datos externa, usualmente en otra máquina remota.

4.5.2 Alternativas

Hay unas cuantas alternativas que cumplen con la misma funcionalidad que Hive y que también deben ser tenidas en cuenta cuando se quiere trabajar en lenguaje SQL.

Una de ellas es **Cloudera Impala**, un motor SQL interactivo que proporciona un mayor rendimiento que Hive haciendo uso de MapReduce. Impala usa un demonio dedicado en cada *datanode* del clúster. Cuando un cliente lanza una consulta, ésta se traspasa a cualquiera de los demonios del clúster que estén corriendo, el coordinador se ocupa de lanzar las subconsultas a los demonios y de recoger los resultados. Así mismo, Impala usa el *metastore* de Hive y soporta los formatos del lenguaje HiveQL, además del

estándar SQL-92, haciendo que resulte muy sencillo migrar entre ambos o tener ambos instalados en el clúster.

Otras alternativas incluyen Presto de Facebook, Apache Drill y Spark SQL. Los dos primeros tienen una arquitectura muy parecida a Impala, mientras que el último hace uso de Spark como motor, permitiendo embeber consultas SQL en los programas Spark. Apache Phoenix sigue otra aproximación diferente, proporcionando SQL en HBASE (sección 4.8)

4.6 Apache Crunch

Apache Crunch [Fou15d] puede ser entendido como un provecto parecido a Apache Pig, si bien en vez de hacer la sintaxis sencilla de cara a los programadores de SQL, la simplifica de cara a los programadores de Java. Los programas se escriben bien en Java o bien en Scala. Tiene la ventaja de poder ofrecer una gran reusabilidad de código, cosa que no es sencilla con los programas de MapReduce directamente, ya que suelen conllevar un grado de especialización de *mappers* y *reducers* alto.

4.7 Apache Zookeeper

Zookeeper [Fou15k] permite a los procesos distribuidos sincronizarse entre ellos a través de un mecanismo de memoria compartida IPC, organizado de forma jerárquica en base al nombre dentro de este espacio de registros de datos. Dentro de la terminología de Zookeeper a estos registros se les denomina znodes y tienen una semejanza con los sistemas de ficheros. Zookeeper está orientado a ofrecer a sus clientes el acceso a registros (*znodes*) con gran ancho de banda, baja latencia, alta disponibilidad y ordenado estrictamente. Está pensado para disponer de mecanismos de replicación de forma que nunca pueda llegar a ser un punto de fallo en grandes despliegues y permite el despliegue de sistemas más complicados de sincronización entre los clientes, que lo usan de base por su sistema de gestión de los registros.

El espacio de nombres que proporciona Zookeeper es como un sistema de ficheros estándar, en el que un nombre se compone de una secuencia de caracteres, configurando un path o ruta completa mediante el separador '/'. Todo registro, *znode*, se identifica por un *path*, y todos los *znodes* tienen un padre, partiendo de la raíz del espacio de nombres (/). Igual que en el caso de los sistema de ficheros, un *znode* no puede ser borrado si tiene algún hijo, sin embargo, aparte de las similitudes, la mayor diferencia de Zookeeper

y un sistema de ficheros es el hecho de que cada *znode* puede tener datos vinculados y cada *znode* tiene un límite con respecto a la cantidad de datos que puede contener. Zookeeper se diseñó para contener información de coordinación: información de estado, configuración, información de localización, etc. Este tipo de meta-información se mide generalmente en bytes o kilobytes como máximo.

El servicio de Zookeeper se replica en varias máquinas del clúster, y estas máquinas mantienen una imagen en memoria compartida de la estructura del árbol junto con los logs de transacciones y un almacenamiento persistente en disco. Debido a que se mantiene en memoria, se consiguen tasas de transferencia muy altas y una latencia muy baja. Los servidores que integran Zookeeper deberán conocerse mútuamente, y mientras una mayoría de ellos estén activos el servicio estará en funcionamiento. Los clientes a su vez también deben conocer la lista de servidores ya que crean un *handle* con el servicio de Zookeeper por medio de esta lista. Los clientes sólo se conectan a uno de los servidores y mantienen una conexión TCP con éste hasta que falle, en cuyo caso se conectan a otro servidor de la lista. En cuanto se conecta un cliente el servidor abre una sesión para el cliente, que será mantenida caso de que se produzca un cambio de servidor. Para realizar operaciones de lectura, o de monitorización/suscripción, el cliente sólo hace uso de un servidor en el acceso a un *znode*, si bien para operaciones de escritura, antes de obtener una respuesta, se debe poner en común entre todos los servidores. La solicitudes de sincronización también son transmitidas a todos los servidores del servicio Zookeeper, pero no requieren de consenso. Esto último hace que la tasa de transferencia aumente para operaciones de lectura en función de un mayor número de servidores, mientras que para peticiones de escritura este índice se decrementa.

El orden es muy importante para Zookeeper, de forma que todas las actualizaciones (update) están estrictamente ordenadas mediante la asignación de un número que lo indica: *zxid* (Zookeeper Transaction Id). Cada *update* tiene un único *zxid*, mientras que las lecturas se ordenan con respecto a los *updates*, haciendo que las respuestas a las lecturas se asocien con el último *zxid* procesado en el servidor que proporciona la lectura.

4.8 Apache HBASE

HBase [Fou15f] es un motor de bases de datos distribuidas orientadas a columnas que se ejecuta encima del sistema de archivos distribuido de Hadoop: HDFS (sección 4.1.1). En general se puede entender HBase como la aplicación de Hadoop a usar cuando se requiere tiempo real para leer o escribir en acceso aleatorio sobre grandes volúmenes de *DataSet*. El concepto de HBase se desarrollo partiendo de buscar escalabilidad lineal vía incorporación de nuevos nodos en el clúster, y si bien HBase no es relacional y no soporta SQL, sí que es capaz de hacer lo que las bases de datos tradicionales o RDBMS no pueden: alojar tablas enormes en nodos de un clúster hecho por hardware *commodity*.

El caso de uso inicial de HBase fue para albergar tablas web, es decir, páginas web recogidas y sus atributos, lo que suponen billones de filas. El análisis por lotes y el parseo mediante MapReduce se entendieron como ejecución de forma continua sobre las tablas web, proporcionando estadísticas y añadiendo nuevas columnas con los textos parseados o datos verificados, pudiendo ser explotados más adelante por un motor de búsqueda. Concurrentemente al proceso anterior, se tenía que dar soporte al acceso aleatorio de los recolectores de webs o *crawlers*, que tenían que actualizar filas aleatoriamente mientras que los resultados se servían a los usuarios en tiempo real.

El projecto HBase comenzó a finales de 2006 por Chad Walters y Jim Kellerman en Powerset, modelado sobre el concepto de Big Table de Google. La primera versión de HBase se liberó como parte de Hadoop 0.15.0 en Octubre de 2007, separándose como proyecto propio de Apache en 2010 y siendo uno de los más usados a nivel mundial.

4.8.1 Conceptos básicos.

Las aplicaciones almacenan los datos en tablas etiquetadas, que consisten en filas y columnas. Las celdas, intersección de fila y columna, están versionadas por defecto con el *timestamp* auto-asignado de HBase en el momento de la inserción de la celda. El contenido de estas es un array de bytes no interpretado, como puedan ser fotos o señales de adquisición binarias. Las claves de las filas son también arrays de bytes, por lo que cualquier cosa puede servir como índice de tabla. Los registros de una tabla se organizan en base a las claves de fila, siendo ordenados en base a los propios bytes. Todas las tablas se acceden a través de su clave primaria. Las columnas se

agrupan por familias de columnas, teniendo todos sus miembros un prefijo común que debe ser compuesto por caracteres imprimibles, aunque el cualificador (*qualifier*) puede ser de cualquier tipo (byte array). El nombre de la familia de columnas y el cualificador se separan mediante el caracter ":". Las familias de columnas deben ser especificadas en el esquema de la creación de la tabla, pero se pueden añadir nuevas familias posteriormente bajo demanda. Físicamente, todas las familias de columnas se almacenan juntas en los sistemas de ficheros, haciendo que HBase sea un almacenamiento orientado a familias de columnas.

Las tablas se organizan automáticamente de forma horizontal en lo que se denominan "regiones", comprendiendo cada una de ellas un subconjunto de filas de la tabla. Una región se denota por la tabla a la que pertenece, su primera fila y su última fila. Inicialmente una tabla comprende una única región, pero mientras la región crece y se traspasa el límite establecido de tamaño, se parte en dos nuevas regiones de aproximadamente igual tamaño. Hasta que se hace el primer corte en la tabla, toda la información que se carga se realiza contra un único nodo, incrementándose el número de nodos conforme se aumenta el número de regiones.

Otro concepto a tener claro es el bloqueo: las actualizaciones de fila son atómicas, sin importar cuantas columnas de la fila se modifiquen se sigue dentro de la misma transacción de fila, haciendo que el modelo de bloqueo sea sencillo.

La implementación sigue los mismos conceptos que HDFS o YARN: clientes, trabajadores y un coordinador. En HBase se tiene un nodo *master* orquestando el clúster de uno o más trabajadores de región. El maestro se ocupa de la preparación del entorno, de la asignación de regiones a los *regionservers* y de la recuperación ante fallos de los anteriores. Se debe denotar que el nodo maestro esta generalmente muy cargado. Los *regionservers* llevan de 0 a N regiones y sirven a las peticiones de lectura de los clientes, coordinando los cortes por región e informando al maestro de HBase sobre las nuevas regiones hijas de forma que pueda manejarse el apagado de los nodos padre y reasignar las nuevas regiones a nodos hijos. Puede verse en la figura 4.6.

HBase depende de Zookeeper (sección 4.7), almacenando información importante para el servicio como la localización del catálogo *hbase:meta* y la dirección del maestro del clúster HBase. La asignación de regiones se realiza por medio de Zookeeper en caso de que los servidores participantes fallen durante el proceso de asignación. La capa de persistencia de HBase viene dada por la API de Hadoop de acceso a ficheros, aunque lo más

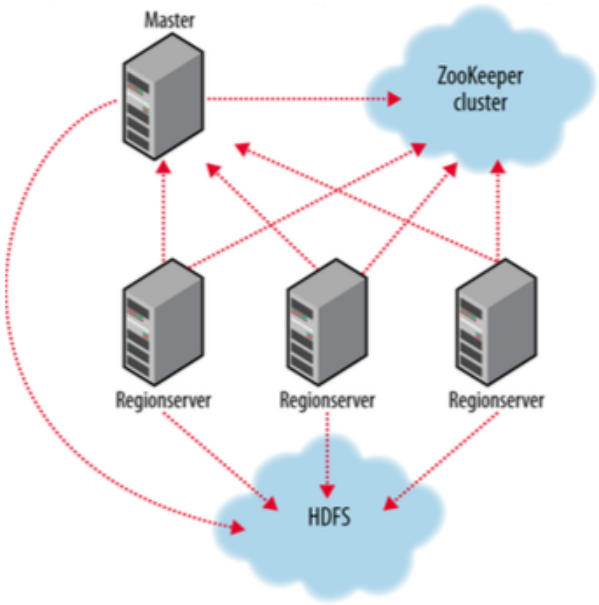


Figura 4.6: Miembros clúster HBase.

común es que la gente ejecute HBase sobre almacenamiento HDFS.

En la tabla catálogo denominada *habse:meta* se mantienen las listas, el estado y las localizaciones de todas las regiones de espacio de usuario que hay en el clúster. Las entradas de esta tabla tienen como clave el nombre de la region, que se compone del nombre de la tabla, de la fila de comienzo de la región, su fecha de creación y un MD5 de todo lo anterior. Teniendo en cuenta que las filas están ordenadas por su clave, encontrar una región que sirva una fila en particular es relativamente directo. Los clientes se conectan a Zookeeper, obtienen la tabla de catálogo y buscan la región para encontrar la localización final de las filas objetivo. La información anterior se cachea en los clientes y la usan mientras siga teniendo validez, caso de que se produzca un error de acceso volverán a consultar a Zookeeper la información ya que ésta ha podido cambiar de emplazamiento bien por fallo del nodo, bien por repartición. Las escrituras que llegan a un *regionserver* son escritas a un log de *commit* y después agregadas a un almacenamiento en memoria, que cuando se llena, vuelca sus contenidos al sistema de ficheros. El log de *commit* también se mapea en HDFS, para evitar pérdidas en caso de fallo de un *regionserver*. Cuando se produce una lectura, lo primero que se busca es el almacenamiento en memoria, y si se puede completar la lectura con la información que existe ahí no se continua el proceso subsiguiente de exploración de los ficheros desde los más nuevos a los más

viejos.

Como proceso de *background* se compactan los ficheros una vez que han llegado al límite establecido, reescribiendo los mismos como un único fichero, ya que contra menos ficheros tenga que leer una consulta habrá un mayor rendimiento. En el proceso de compactación se limpian las versiones acorde al máximo establecido en el esquema, así como se eliminan las filas borradas y las celdas caducadas. Un proceso separado monitoriza los tamaños de los ficheros volcados de memoria, haciendo el particionamiento de los mismos acorde a los máximos configurados.

4.8.2 **HBase vs RDBMS.**

HBase es un sistema de almacenamiento de datos distribuido orientado a columnas, diseñado desde el principio para la escalabilidad en todas direcciones: crecer en número de filas (billones) y en número de columnas (millones). Es capaz de particionarse y replicarse en clústers de miles de nodos de forma automática. Los esquemas de tabla son imágenes del almacenamiento físico, creando un sistema para serialización, almacenamiento y recuperación de datos muy eficiente.

Un sistema de base de datos tradicional o RDBMs, sigue las 12 reglas de Codd [Cod90]: tienen un esquema fijo, son bases de datos orientadas a las filas con propiedades ACID y tienen un motor de búsquedas muy avanzado en lenguaje SQL. El punto fuerte de estos sistemas está en la consistencia de los datos, la integridad referencial, la abstracción del medio físico y la disposición de un lenguaje, con su correspondiente motor e intérprete, muy potente para la realización de búsquedas complejas.

Para la mayor parte de las aplicaciones de pequeño o gran volumen, no hay un substituto para los sistemas RDBMs, pero si se debe escalar en el tamaño de los datos, la concurrencia de escrituras/lecturas, o ambos; rápidamente el desarrollador o el arquitecto se dan cuenta de que las bases de datos tradicionales no cumplen con los requisitos esperados, rompiendo en estos sistemas las restricciones ACID, y o trampeando los sistemas RDBMs de forma no consistente para abordar la situación, punto en el que los sistemas como HBase son fuertes.

4.9 Apache Kafka

Apache Kafka [Fou15h] es un proyecto open-source que proporciona un servicio de mensajería basado en el modelo productor-consumidor, ofreciendo funcionalidades avanzadas para sistemas distribuidos y con replicación. Kafka es una solución orientada a sistemas de tiempo-real, que puede transferir la información que se recibe a múltiples consumidores para poder evaluarla a rápidamente. Kafka proporciona una capa de desacoplamiento entre productores y consumidores, pudiendo escalarse de la mejor forma posible para dimensionar ambos lados, productores y consumidores, acorde a sus requisitos temporales de generación de información y de consumo de la misma.

Ofrece las siguientes características:

- Persistencia de capa de mensajes. Para conseguir que el análisis de Big Data sea efectivo no se puede permitir la pérdida de ningún tipo de información, de forma que Apache Kafka ha sido diseñado con estructuras de disco $O(1)$ que proporcionan un rendimiento constante en el tiempo incluso con grandes volúmenes de mensajes almacenados (TBs).

- Alto ancho de banda. Kafka está diseñado para trabajar con *Commodity Hardware* y soportar cientos de MBs de lecturas y escrituras por segundo de un gran número de clientes.

- Sistema distribuido. El desarrollo del proyecto Apache Kafka está centrado en clústers y proporciona un correcto particionado de la información, así como una capa de distribución de la información para los consumidores, siempre manteniendo el orden de forma estricta. Además el clúster puede crecer de forma elástica y transparente sin latencias en el servicio.

- Soporte multicliente. Se proporciona una integración sencilla con numerosos tipos de clientes: Java, .NET, PHP, Ruby, Python y C/C++.

- Orientado a tiempo real. Los mensajes que se producen pueden ser visibles inmediatamente por los consumidores, característica usada en sistemas basados en eventos tipo CEP (*Complex Event Processing*).

La imagen 4.7 muestra un diagrama típico de Apache Kafka para infraestructuras Big Data, donde se realizará la agregación y el análisis de la información recabada.

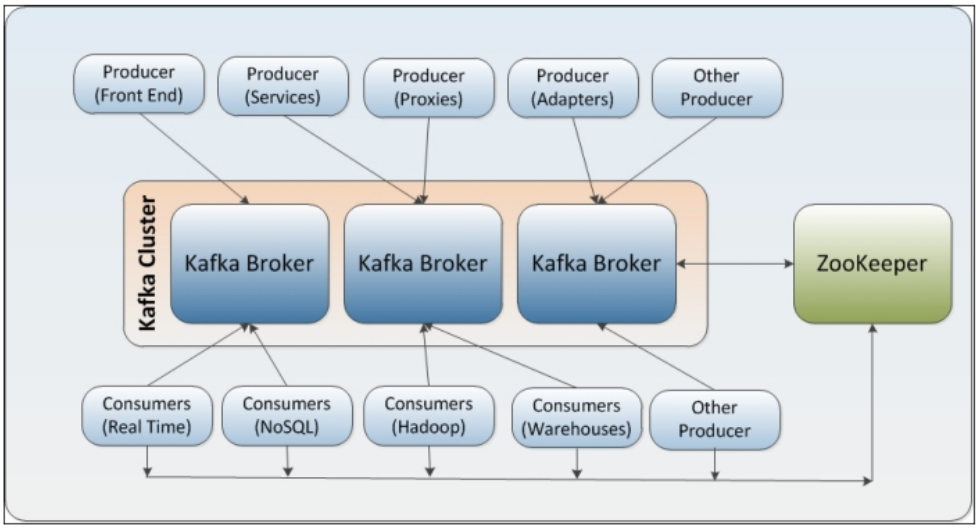


Figura 4.7: Apache Kafka - Escenario Típico.

En la imagen 4.7 se muestran diferentes tipos de productores de información, como los siguientes:

- Aplicaciones Web, *Frontends*, generando logs de aplicación.
- Proxies productivos generando logs para *web-analytics*.
- *Producer adapters* generando logs de transformación.
- *Producer services* generando logs con las llamadas correspondientes a modo de traza del servicio.

En el lado de los consumidores, también puede haber un número diverso de tipos, a modo de ejemplo:

- Consumidores *off-line*, que realizan el parseo de los mensajes y la historificación de los mismos en Hadoop o en sistemas de *Data Wharehouse* para análisis de la información mediante herramientas ETL.
- Consumidores en tiempo casi real que procesan los mensajes y realizan el almacenaje de los mismos en sistemas tipo NoSQL, como HBASE [Fou15f] o Cassandra [Fou15l], para permitir el análisis de la información en tiempo real.

- Consumidores en tiempo real, como Spark [Fou15a] o Storm [Fou15j], que filtran los mensajes en memoria y lanzan alertas o realizan funciones de algoritmia compleja para dar soluciones avanzadas en base a la información.

Apache Kafka trata de unificar el procesamiento *on-line* y *off-line*, proveyendo con mecanismos de carga en paralelo en sistemas Hadoop, así como proporcionando mecanismos para el consumo en tiempo real de la información organizada en particiones dentro de un clúster de máquinas. Kafka se puede comparar con Scribe o Flume en su funcionalidad de procesado de la información sobre flujos de datos (*streaming*); pero desde un punto de vista arquitectónico, está más cerca de sistemas de mensaje de colas tradicionales como ActiveMQ o RabitMQ.

Hay muchos casos de uso para Apache Kafka, pero principalmente se está usando en el mercado para:

1. Agregación de logs. Este es el proceso de recogida de ficheros de logs de múltiples servidores y reunirlos en un único sistema central (por ejemplo: un sistema de ficheros basado en HDFS) desde el que pueda procesarse toda la información en su conjunto. El uso de Apache Kafka para este tipo de soluciones proporciona una abstracción de los logs o eventos a un flujo de mensajes, eliminando las dependencias de los ficheros en cuestión. Se proporciona a su vez un procesamiento muy potente que soporta el análisis de múltiples fuentes de información y consumo de datos distribuidos.

2. Procesamiento de flujos (Streams). Kafka se puede usar como herramienta donde la información recogida deba ser procesada y completada o transformada en nuevos mensajes/eventos antes de ser susceptible de ser analizada. Este proceso se suele denominar como *Streaming* o *Stream processing*.

3. *Commit Logs*. Kafka se puede usar para recoger logs en sistemas distribuidos de cualquier magnitud, permitiendo que la replicación que ofrece Kafka pueda recuperar cualquiera de los nodos en caso de fallo.

4. *Stream Tracking*. Una funcionalidad muy importante de Kafka es la captura de datos de usuario como número de visitas por página, búsquedas, etc, al igual que *feeds* en tiempo real. Esta información se publica en un topic central, en la que puede haber múltiples consumidores de la información con diferente tipo de tecnologías entre las que se incluyen procesamiento en tiempo real y monitorización.

5. Mensajería. Los *brokers* de mensajes se usan para desacoplar el procesamiento de datos de los productores de datos. Kafka suele reemplazar a muchos de los brokers que se estaban usando, ofreciendo mejor transferencia, replicación, tolerancia a fallos y particionamiento de la información.

Algunos ejemplos de compañías que están haciendo uso de Apache Kafka: LinkedIn, DataShift, Twitter, Foursquare, Square, etc.

4.9.1 Arquitectura y diseño de Apache Kafka

Debido a los sobrecostes asociados con JMS, sus varias implementaciones y sus respectivas limitaciones a la hora de realizar un correcto escalado, LinkedIn [Lin15] decidió crear Apache Kafka para solventar su necesidad de monitorización de flujos de información y métricas operativas como CPU, uso de E/S y tiempos de procesamiento. Durante el desarrollo de Kafka, el foco principal estaba en proporcionar:

- Una API para productores y consumidores que permitiese una implementación personalizada.
- Bajos costes de red y almacenamiento, pero que permitiese la persistencia en disco.
- Una gran transferencia para permitir procesar millones de mensajes, tanto de publicación como de suscripción.
- Una arquitectura distribuida y altamente escalable para soportar la entrega de mensajes con muy baja latencia.
- Auto balanceo de múltiples consumidores en caso de fallo.
- Tolerancia a fallos garantizada en caso de fallo de servidores.

Kafka no se concibió ni como una plataforma de encolado de mensajes donde los mismos son recibidos por el mismo consumidor o por el mismo pool de consumidores, ni como una plataforma de productor-consumidor donde los mensajes sean publicados a todos los clientes. De forma muy básica, un productor genera un mensaje que se envía a un tópico de Kafka. Este tópico puede ser considerado como la categoría del mensaje, el nombre del *feed* donde los mensajes son publicados. Los tópicos de Kafka son creados

dentro de un broker, que actúa como servidor de Kafka y que también puede almacenar los mensajes si es requerido. Los consumidores se suscriben a un tópico de Kafka (a uno o a varios tópicos) para recoger los mensajes que vayan llegando.

Los brokers y los consumidores hacen uso de Zookeeper para obtener la información de estado y poder saber el *offset* correspondiente de los mensajes.

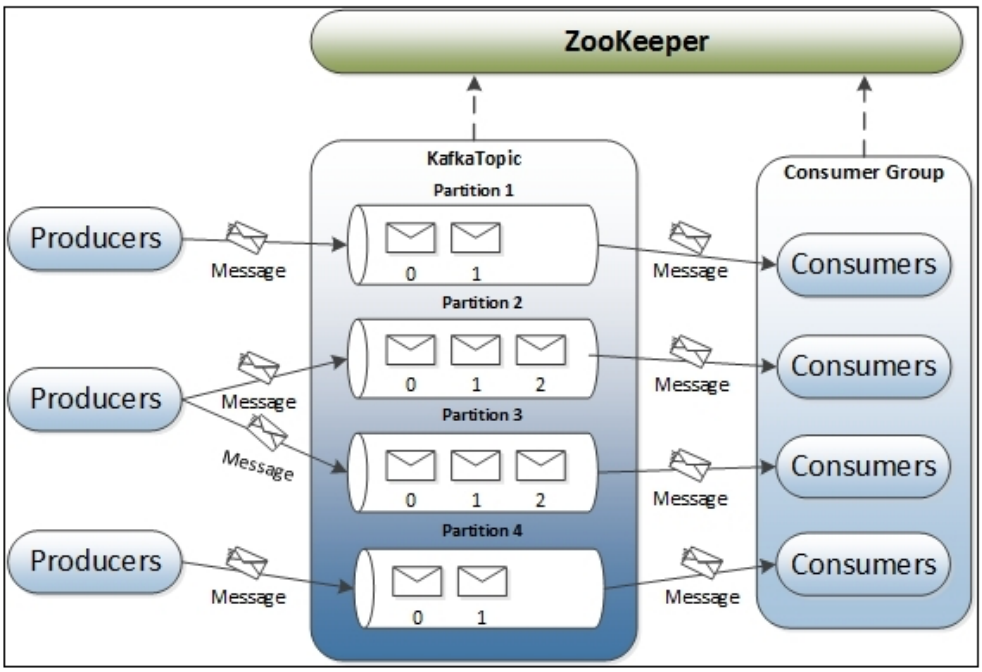


Figura 4.8: Apache Kafka - Arquitectura.

En la figura 4.8 se puede apreciar una arquitectura de nodo único con un sólo broker, pero en el que se despliegan cuatro particiones. Dentro de un mismo tópico, cada partición es mapeada a un fichero lógico de log, que representa como un conjunto de segmentos de ficheros con el mismo tamaño. Cada partición es una secuencia de mensajes ordenada e inmutable, en la que cuando un nuevo mensaje es publicado en la partición, el broker añade el mensaje al último segmento de fichero. Estos segmentos de ficheros se vuelcan a disco tras la llegada de un número configurable de mensajes o cuando ha transcurrido un determinado intervalo de tiempo. Una vez que un segmento de fichero ha sido transferido a disco quedará disponible para los consumidores.

Todas las particiones de los mensajes son asignadas con un número úni-

co secuencial, denominado *offset*. Este *offset* se usa para identificar cada mensaje dentro de cada partición. Así mismo, una partición puede ser replicada a un número configurable de servidores para la tolerancia a fallos. Cada partición disponible en cualquiera de los servidores actúa como el líder y puede tener un número de seguidores que varía entre 0 y N. En este caso, el líder es responsable de soportar todas las peticiones de lectura y escritura de la partición, mientras que los seguidores replican la información de forma asíncrona desde el líder. Kafka mantiene dinámicamente un conjunto de ISR (In-Sync Replicas), que se recogen del líder y se persisten en Zookeeper, de forma que si el líder cae, uno de los seguidores se convertirá automáticamente en el nuevo líder. Dentro un clúster Kafka, cada servidor juega dos roles: actúa como líder para alguna de sus particiones y es seguidor de otras particiones, siendo esta dualidad la que permite el reparto de la carga dentro del clúster.

Un consumidor se entiende y representado como un proceso, y estos procesos se organizan dentro de grupos denominados *Consumer Groups*. En general, un mensaje de un tópico es consumido por un único proceso de un *Consumer Group*, pero para que ese mismo mensaje puede ser consumido por varios consumidores, estos consumidores deberán pertenecer a distintos *Consumer Groups*. Los consumidores siempre consumen mensajes de una partición en particular secuencialmente y mantienen el *offset* correspondiente. Los consumidores pueden pedir al *broker* de forma asíncrona el tamaño del buffer que requieren para obtener todos los mensajes pendientes desde un *offset* concreto.

En el diseño de Kafka los *brokers* no tienen estado, con lo que el estado del mensaje de cualquier mensaje consumido deberá llevarlo el propio consumidor, mientras que Kafka no se preocupa de qué ha sido consumido ni por quién. Esto conlleva que si no se realiza una correcta implementación del consumidor, un mensaje puede ser leído múltiples veces. Kafka define un SLA (*Service Level Agreement*) basado en el tiempo, y en línea con este SLA, un mensaje será borrado si ha sido mantenido en el *broker* por más tiempo del indicado en la política anterior.

Kafka proporciona las siguientes semánticas entre un productor y un consumidor:

- Los mensajes nunca son retransmitidos pero pueden ser perdidos.
- Los mensajes pueden ser retransmitidos pero nunca perdidos.
- Los mensajes se entregan una vez y sólo una.

Cuando se publica un mensaje es transferido al log, si bien, si un productor experimenta un problema de red mientras se está publicando, no se puede estar seguro de si el error ha pasado antes o después de que el mensaje haya sido publicado. Una vez publicado, el mensaje no será borrado mientras haya algún *broker* presente que tenga réplica de la partición correspondiente. Para evitar esta situación con el productor existen configuraciones que permiten garantizar la entrega.

Desde el punto de vista del consumidor, las réplicas tienen exactamente el mismo log con los mismos *offsets*, y es el consumidor quien controla su posición dentro del log. Kafka asegura a los consumidores que el mensaje será entregado al menos una vez mediante: lectura del mensaje, procesamiento del mismo y salvado de la posición correspondiente. Caso de que el proceso de consumidor falle después del procesamiento del mensaje, pero antes del salvado del offset, otro consumidor podrá volver a reprocesar mensajes que ya estaban procesados.

4.9.2 **Compactación de logs.**

Este mecanismo garantiza que el último valor para cada clave de mensaje dentro de un log de una partición de un topic será mantenido y entregado mediante el borrado de registros con la misma clave de mensaje. Este mecanismo permite la solución a determinados fallos, así como los reinicios de los sistemas.

En un clúster Kafka, la política de retención para un tópico se puede fijar en base al tamaño, al tiempo, o en base a la compactación de logs. Esta última garantiza lo siguiente:

1. El orden de los mensajes se mantiene siempre.

2. Los mensajes tienen *offsets* secuenciales y el *offset* nunca cambia.

3. Las lecturas desde el *offset* 0, o el procesado de un consumidor desde el comienzo del log, verán siempre por lo menos el último valor de los registros en el orden en el que fueron escritos.

La compactación de logs se lleva a cabo por un *pool* de procesos en *background* que vuelven a copiar los segmentos de logs borrando los registros que aparecen en la parte superior del log en cuestión. Los siguientes puntos resumen los características más importantes de Apache Kafka:

- El *backbone* fundamental de Kafka es el cacheado de mensajes y el almacenamiento de los mismos en los sistemas de ficheros. En Kafka, los datos son inmediatamente escritos a una página del Kernel del sistema operativo. El cacheado y el volcado a disco de los datos son configurables.

- Kafka proporciona una retención más larga de mensajes después de su consumo, permitiendo a los consumidores volver a procesar si es necesario.

- Kafka usa un conjunto de mensajes para agrupar los mensajes haciendo menor la sobrecarga de la red.

- En Kafka el estado de los mensajes consumidos se mantiene a nivel del consumidor, lo que puede conllevar dos pegas: pérdida de mensajes debido a problemas, múltiples entregas del mismo mensaje. En general, los consumidores guardan el estado en Zookeeper, pero Kafka también permite el salvado de esta información en otros tipos de sistemas de almacenamiento que se usan para OLTP (*Online Transaction Processing*).

- Los productores y consumidores trabajan bajo el paradigma tradicional de *push-and-pull*, donde los productores envían el mensaje al *broker* de Kafka y los consumidores recogen el mensaje del *broker*.

- Kafka carece del concepto de maestro y trata a todos los brokers como nodos. Esto facilita enormemente la adición o el borrado de un *broker* de Kafka en cualquier momento, dado que la información se mantiene en Zookeeper y es compartida entre todos los consumidores.

- Los productores también pueden elegir si desean el envío de mensajes síncrono o asíncrono al *broker*.

4.9.3 **Compresión de mensajes en Kafka.**

Para aquellos casos en los que la transferencia por red sea un cuello de botella, Kafka proporciona compresión para la entrega de mensajes de forma eficiente. Kafka soporta compresión eficiente permitiendo la compresión de múltiples mensajes juntos, en vez de comprimir cada mensaje individualmente. En Kafka, los datos se comprimen por el productor de los mismos, pudiendo usar GZIP o Snappy como protocolos de compresión. Los

Cuadro 4.2: Propiedades de compresión de un broker.

Nombre de la propiedad	Descripción
compression.codec	Este parámetro especifica el codec de compresión para todos los datos generados por el productor. Los valores válidos son: none, gzip y snappy. Defecto: none
compressed.topics	Este parámetro permite fijar si la compresión debe ser encendida para determinados tópicos. Si la compresión está fijada a otra cosa que no sea None, se habilita la compresión para los tópicos especificados. Si la lista de tópicos está vacía, se establece la compresión para todos los tópicos. Si la compresión es none, la compresión es deshabilitada para todos los tópicos. Defecto: null

siguientes parámetros deberán ser proporcionados para usar la compresión en el lado del productor:

La clase *ByteBufferMessageSet* que representa un conjunto de mensajes, puede contener datos tanto sin comprimir como comprimidos. Para diferenciar entre mensajes comprimidos y no comprimidos, se añade un byte de atributos de compresión en la cabecera del mensaje.

4.9.4 **Replicación en Kafka.**

Para comenzar, lo primero es explicar la estrategia de particionado que se usa en los *brokers*. La decisión de cómo debe particionarse un mensaje se toma en el productor, siendo el broker quien mantiene y almacena los mensajes en el orden en el que llegan. El número de particiones puede ser configurado para cada tópico en el ámbito del *Broker* de Kafka. La replicación de Kafka se ha diseñado para que los mensajes sean publicados y consumidos incluso en el caso de fallo de los *brokers*. Tanto los productores como los consumidores son agnósticos con respecto a la replicación en Kafka, lo que se explica en la figura 4.9.

En la replicación, cada partición de un mensaje tiene n réplicas y puede soportar $n-1$ fallos para garantizar la entrega de los mensajes. Dentro de

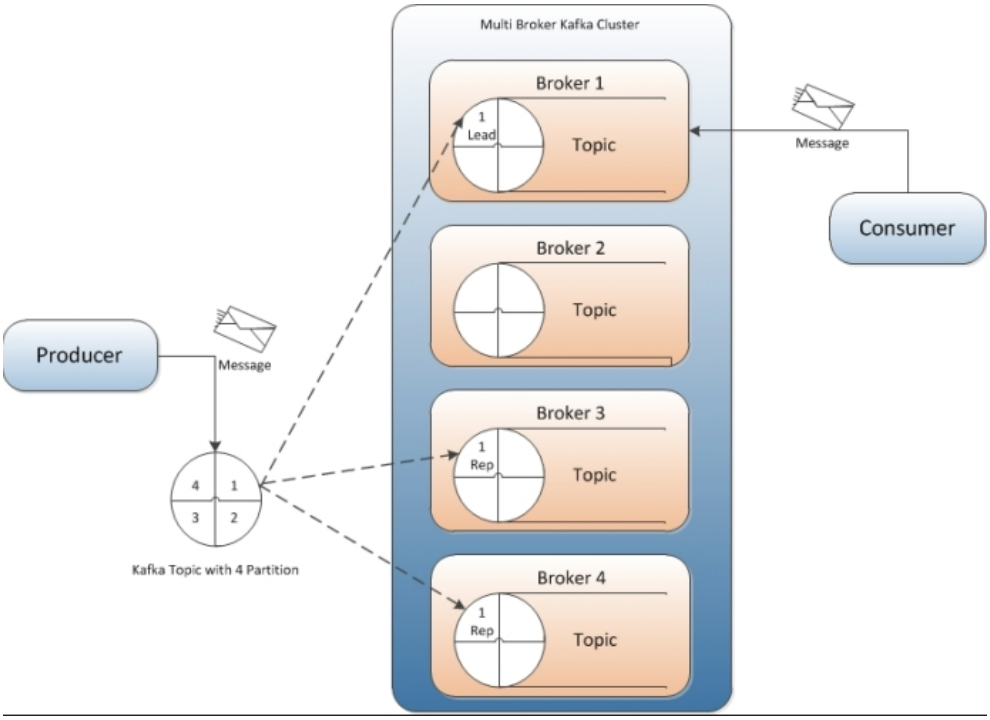


Figura 4.9: Apache Kafka - Replicación.

las n réplicas, una réplica actual como líder para el resto, información que se mantiene en Zookeeper dentro del ISR (*In-Sync Replicas*). Cada réplica almacena su parte del mensaje en los logs y *offsets* locales y es periódicamente sincronizado a disco. Este proceso también se asegura de que cada mensaje sea escrito a todas las réplicas o a ninguna de ellas. Kafka soporta los siguientes modos de replicación:

- Replicación síncrona. El productor localiza la réplica líder desde la información de Zookeeper y después publica el mensaje. Tan pronto como el mensaje es publicado, se escribe en el log de la réplica líder y todos los seguidores de ella copian el mensaje. Debido a que se usa un único canal, el orden de los mensajes es asegurado. Cada seguidor manda un *acknowledge* a la réplica líder una vez de que el mensaje es escrito al log correspondiente. Una vez que las réplicas están completas y se reciben todos los *acknowledges*, la réplica líder manda esta información en forma de ACK al productor. Desde el lado del consumidor todos los mensajes son consumidos desde la réplica líder.

- Replicación Asíncrona. La única diferencia con este modo es que tan

pronto como la réplica líder escribe el mensaje en el log, se manda el ACK al productor y no se espera a que se tengan los ACKs de las réplicas seguidoras, si bien este modelo no asegura la entrega de los mensajes en caso de un fallo de un *broker*.

Si alguno de los seguidores falla, el líder quita de su lista de seguidores a este proceso. Cuando el nodo que ha fallado vuelve, trunca su log al último punto donde la réplica estaba correcta y comienza a recoger todos los mensajes desde el líder. Tan pronto como el seguidor esté completamente sincronizado, el líder volverá a incluirlo en su lista de ISR.

Si la réplica líder falla, bien durante la escritura del mensaje a la partición de su log local, bien antes de mandar el ACK al productor, la partición del mensaje se vuelve a enviar por el productor al nuevo líder. El proceso de elección del nuevo líder implica que todos los seguidores se registren en Zookeeper. La primera réplica que se registra en Zookeeper se convierte en el nuevo líder y su offset del log, LEO (*Log End Offset*) se convierte en el offset del último mensaje procesado, HW (*High Watermark*). El resto de las réplicas registradas se convierten en seguidores del nuevo líder elegido y cada réplica registra un listener en Zookeeper que le notificará cuando haya un cambio de líder. Cada vez que un nuevo líder es elegido y se le notifica a la réplica que no lo es, esta última trunca el log al offset del último mensaje subido correctamente y comienza a recoger los mensajes del nuevo líder. El nuevo líder espera bien un determinado tiempo o bien a que todas las réplicas estén activas y sincronizadas, antes de escribir la lista ISR en Zookeeper. Una vez se confirma el proceso anterior, permitirá que se conecten para lectura y escritura los clientes.

La replicación en Kafka asegura durabilidad, fortaleza y alta disponibilidad. Se garantiza que cualquier mensaje satisfactoriamente publicado no será perdido hasta que sea consumido, incluso con el fallo de *brokers*.

4.9.4.1 **Integración de Kafka con Hadoop**

La siguiente imagen 4.10 muestra un esquema general de la arquitectura de Hadoop y su integración con Kafka.

Hadoop tiene los siguientes componentes principales:

- *NameNode*: Este es el punto principal de la interacción con el sistema de archivos HDFS. El namenode almacena información sobre los bloques de la información y cómo están distribuidos a lo largo del clúster.

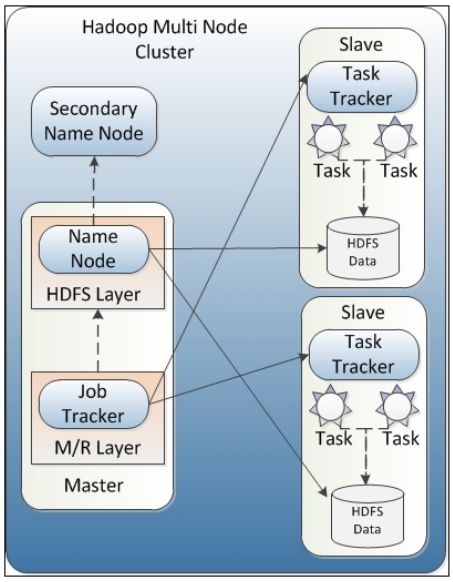


Figura 4.10: Apache Kafka - Integración Hadoop.

- *Secondary NameNode*. Este nodo almacena los logs de ediciones, permitiendo la restauración de la última información de estado del sistema HDFS en caso de que un *namenode* falle.
- *DataNode*: Estos nodos son donde reside realmente la información, que se almacena en bloques y que son replicados en otros DataNodes.
- *Job Tracker*. Este proceso es el responsable de partir los trabajos de MapReduce en tareas más pequeñas.
- *Task Tracker*. Este proceso es el encargado de la ejecución de las tareas que han sido creadas por el *Job Tracker*.

Un productor de Kafka para Hadoop proporciona un puente para publicar datos desde un clúster de Hadoop a Kafka. Para un productor de Kafka, los tópicos se consideran como URIs y para conectarse a un *broker* determinado, las URIs tienen el siguiente formato: *kafka://<kafka-broker>/<kafka-topic>*

El productor de Hadoop proporciona dos posibles alternativas para recoger información desde Hadoop:

1. Haciendo uso de un script en Pig y escribiendo los mensajes en formato Avro. En esta aproximación, los productores de Kafka usan scripts de Pig para escribir información en formato binario mediante Avro,

donde cada fila representa un mensaje. Para el envío de la información a un clúster Kafka, la clase AvroKafkaStorage se ocupa de formatear el mensaje acorde al esquema definido para Kafka de Avro, y se conecta con la URI indicada. Para poder hacer uso de esta funcionalidad se deberán registrar convenientemente los JARs de Kafka.

2. Usando la class *OutputFormat* de Kafka como salida de los Jobs de Hadoop. Esta clase, que hereda de la clase *OutputFormat* de Hadoop, se usa para publicar la información a un clúster Kafka.

Un consumidor de Kafka para Hadoop no es más que un *job* que se ocupa de recoger la información de un broker de Kafka y lo almacena en el sistema de archivos distribuidos de HDFS. Este trabajo se ocupa de realizar una carga en paralelo desde Kafka a HDFS, y el número de mappers para la carga de los datos dependerá del número de ficheros que existan en el directorio de entrada. El directorio de salida contendrá los datos provenientes del clúster Kafka y la actualización de los mensajes recibidos. Los propios *mappers* se ocupan de escribir el último *offset* consumido en el sistema HDFS al final de cada tarea de *map*, por lo que si un *job* falla y se reinicia, cada *mapper* sólo deberá reiniciar desde el último *offset* guardado en el sistema de HDFS.

4.10 Apache Spark.

El proyecto Apache Spark [Fou15a] es un framework de computación para clústers de máquinas diseñado para el procesamiento de gran volumen de datos. Spark, al contrario que la mayoría de los frameworks explicados en este capítulo, no hace uso del framework *MapReduce* como motor de ejecución. Spark hace uso de su propio *runtime* para la ejecución de las tareas en el clúster. El proyecto se integra perfectamente con Hadoop y puede correr encima de YARN, de Apache Mesos o por sí mismo en las máquinas del clúster, siendo capaz de leer ficheros en formato Hadoop y hace uso de la capa de almacenamiento distribuido del entorno Hadoop: HDFS (sección 4.1.1).

Spark mejora el rendimiento que se puede obtener en las mismas tareas que MapReduce haciendo uso de la memoria del clúster, ya que almacena en ésta los grandes volúmenes de datos entre trabajos. Las aplicaciones que más se benefician de este tipo de funcionamiento son aquellas que usan algoritmia iterativa y aquellas que requieren análisis interactivos.

Spark ofrece un entorno con una API muy rica en funcionalidades en tres lenguajes: Python, Scala y Java; habiéndose desarrollado en el segundo de ellos, Scala [Sca15], que es donde se encuentran las mayores bondades de la plataforma. Así mismo, cuenta con una *shell* REPL (*read-eval-print loop*) tanto para Scala como para Python, facilitando las tareas de desarrollo y/o análisis de datos iniciales para exploración del *DataSet*.

Spark proporciona una buena plataforma sobre la que desarrollar herramientas analíticas, ofreciendo además otros módulos de funcionalidad avanzada:

- MLlib: Librería de Machine Learning.
- GraphX: Librería de procesamiento de grafos.
- Spark Streaming (sección 4.10.3): librería para el procesamiento de flujos.
- Spark SQL: Soporte para SQL.

Spark, al igual que MapReduce, tiene el concepto de trabajo, aunque en este caso es más general ya que un trabajo se compone de un grafo acíclico dirigido (DAG) arbitrario compuesto de etapas, siendo cada etapa un equivalente a una fase de *Map* o de *Reduce*. Cada etapa a su vez se divide en tareas, que se ejecutan de forma paralela sobre particiones del los RDDs distribuidos a lo largo del clúster. Un RDD (*Resilient Distributed Dataset*) no es más que una abstracción que usa el framework para una colección de objetos que se particiona en varias máquinas dentro de un clúster, que sufren una serie de transformaciones y de acciones a través de la lógica de las aplicaciones. El termino de *resilient* viene dado por el hecho de que Spark puede reconstruir automáticamente una partición perdida usando la herencia de transformaciones/acciones hechas sobre el RDD original, concepto que se denomina *linkage*.

Un trabajo siempre corre en el contexto de una aplicación (*SparkContext()*), pudiendo ejecutar uno o más trabajos, en serie o en paralelo, y proporciona mecanismos para acceder a RDDs que hayan sido almacenados en memoria (*cached*) en otro trabajo de la misma aplicación.

4.10.1 **Resilient Distributed Datasets.**

La creación de los RDD, ya que son el corazón de todos los programas en Spark, se puede realizar: desde una colección de objetos, también deno-

minado "paralelización de una colección"; desde un *DataSet* de almacenamiento externo, como pueda ser HDFS, o por una transformación de un RDD existente.

Spark proporciona dos categorías de trabajo con los RDDs: transformaciones y acciones. La diferencia entre ellas es que una transformación genera como salida un RDD nuevo a partir de un RDD existente, mientras que las acciones se ocupan lanzar cálculos sobre un RDD haciendo alguna operación con los datos. En las acciones se incluyen aquellas que permiten mostrar datos por pantalla o el salvado de los mismos a un sistema de almacenamiento externo. Otra diferencia entre ambas categorías es que las acciones tienen efecto inmediato, mientras que las transformaciones no (*lazy computing*), realizándose únicamente cuando una acción se solicita sobre los datos transformados. La librería de Spark contiene un conjunto de operaciones para transformación muy rico: mappeo, agrupación, agregación, repartición, sampleo y unión de RDDs; incluso hay transformaciones que generan colecciones o cálculos estadísticos sobre RDDs, y salvado de RDDs a disco. Las transformaciones de agregación más importantes son: *reduceByKey()*, *foldByKey() y aggregateByKey()*.

Una funcionalidad muy apropiada y versátil de Spark es la persistencia, mediante la función de *cache()*, aunque no se realiza la operación instantáneamente, se marca para que cuando se calcule el RDD correspondiente se almacene en memoria. El uso de esta funcionalidad de forma adecuada hace que el tiempo de ejecución se reduzca ampliamente al procesar *DataSets* de gran volumen. De igual forma, cuando se requiere una colección de valores calculada puede ser traspasada a la memoria de todos los ejecutores del clúster mediante la orden *broadcast()*.

4.10.2 **Spark Jobs y Gestores del clúster.**

El lanzamiento de trabajos en Spark conlleva dos entidades importantes: el *driver* y los *executors*. El primero es quien tiene el contexto de ejecución de la aplicación y organiza las tareas para un determinado trabajo. El segundo está en ejecución para realizar las tareas que indique el *driver* en la aplicación. Un trabajo se lanza automáticamente cuando una acción se realiza en un RDD, que internamente hace se que lance un *runJob()* en el contexto de la aplicación, siendo el contexto el que pasa la llamada al planificador que se ejecuta como parte del *driver*. El planificador se compone de dos partes: un planificador DAG que trocea el trabajo en etapas DAG, y un planificador de tareas que es responsable de introducir las tareas de cada etapa al

clúster.

Para entender como un trabajo se descompone en etapas, se debe saber que puede haber dos tipos de tareas en una etapa: *shuffle* y *result*. Las primeras son equiparables a un *Map* de MapReduce. Cada tarea S de *map* ejecuta una computación sobre una partición del RDD, escribiendo los resultados a un nuevo conjunto de particiones, que serán recuperadas en una etapa posterior. El tipo S de tareas siempre se encuentran en todas las etapas menos en la etapa final. Las segundas, tareas R, sólo corren en la tarea final, devolviendo el resultado al programa de usuario. Cada tarea R ejecuta el cálculo sobre una partición del RDD, enviando después el mismo al *driver*, siendo este último quien recopila los resultados de cada partición para obtener el resultado final.

Spark para poder gestionar el lanzamiento de tareas puede usar uno de los siguientes gestores de clúster:

- Modo *local*. En este modo sólo hay un ejecutor en funcionamiento, corriendo en la misma máquina JVM que el *driver*. Este modo generalmente es el que se usa para desarrollo, pruebas o análisis de una pequeña porción del *DataSet*. La URL que se especifica para el *master* es: local (1 sólo hilo), local[*] (un hilo por cada *core* de la máquina de ejecución), local [n] (n hilos).

- Modo *Standalone*. Es una implementación simple de ejecutar un único *master* de Spark en el clúster haciendo el resto de máquinas *workers* del mismo. Cuando se lanza una aplicación en este modo, el *master* solicita a los *workers* que lancen procesos de ejecución a favor de la aplicación que se lanza. La URL para este tipo de clúster es *spark://host:port*.

- Modo Apache Mesos. Apache Mesos es un gestor de recursos de clústers multi-propósito que permite una compartición de los mismos muy bien gestionada. Por defecto, cada tarea de Spark en este tipo de gestor se ejecuta como una tarea Mesos, haciendo que se usen los recursos de la forma más eficiente posible con la contrapartida del sobrecoste en la penalización del lanzamiento. La URL para este tipo de clústers es *mesos://host:port*.

- Modo YARN. Como se ha explicado con anterioridad, sección 4.1.2, éste es el gestor de recursos que se usa en Hadoop. Cada aplicación de Spark corresponde con una instancia de una aplicación YARN, y cada

ejecutor se lanza en su propio contenedor YARN. Las URLS para lanzar este tipo de clúster son parametrizadas: *yarn-client*, figura 4.11; ó *yarn-clúster*, figura 4.12. El primero de los tipos de URLs se usa para aplicaciones que requieran de un componente interactivo, como por ejemplo la *shell* REPL, haciendo su uso muy extendido en etapas iniciales de depuración ya que hacen visibles los resultados rápidamente. El segundo tipo, *yarn-clúster* se usa para trabajos productivos, que hace que la aplicación corra íntegramente dentro del clúster, almacenando los logs de ejecución para inspecciones posteriores.

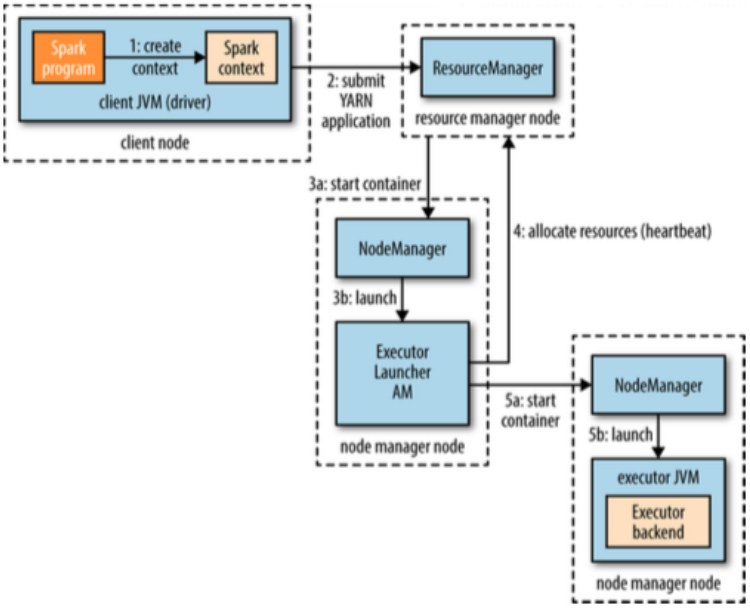


Figura 4.11: Spark en YARN, modo cliente.

4.10.3 Spark Streaming.

Son muchas las aplicaciones que se pueden beneficiar de procesar la información tan pronto como llega. *Spark Streaming* es un módulo de Spark que sigue basándose en el concepto de RDD, sección 4.10.1, pero lo abstrae a lo que se llama *streams discretizados* ó *DStreams*. Un *DStream* es una secuencia de datos llegados sobre un periodo de tiempo. Internamente, un *DStream* no es más que una secuencia de RDDs llegados en un determinado tiempo. Los *DStreams* pueden ser creados desde varias fuentes, entre las que se incluyen: Flume, sección 4.2; Kafka, sección 4.9; o HDFS, sección 4.1.1. Una vez construidos ofrecen dos posibles operaciones:

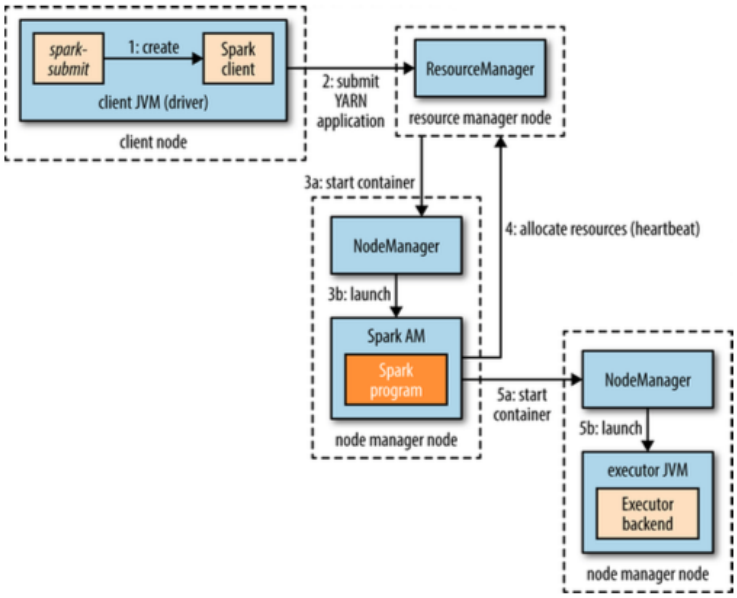


Figura 4.12: Spark en YARN, modo clúster.

- Transformaciones: que son aquellas operaciones que devuelven un nuevo *DStream*.
- Operaciones de salida (*output operations*): que escriben los resultados a sistemas externos.

Los *DStreams* ofrecen muchas de las operaciones que se tienen disponibles en los RDDs, además de ofrecer operaciones con relación a la temporalidad, como las ventanas deslizantes. Para lograr una disponibilidad de 24x7, las aplicaciones que usen *Spark Streaming* requieren de una configuración adicional, que en Spark se denomina *checkpointing*, permitiendo almacenar información en un sistema de ficheros fiable como HDFS.

Spark Streaming usa una arquitectura denominada de "micro-lotes", donde el flujo de computación se trata como una serie continua de cálculos por lotes muy pequeños. Los datos se pueden recibir de numerosas fuentes y son agrupados en pequeños lotes, que se crean a intervalos temporales regulares. Al principio de cada intervalo de tiempo se crea un nuevo lote, donde cualquier dato que llegue en ese intervalo será introducido en el mismo. Al final del intervalo de tiempo se finaliza la inserción de nuevos datos en ese "micro-lote". El intervalo del "micro-lote" se especifica en un parámetro denominado *batch interval* y típicamente suele ser de entre unos 500 ms a varios segundos, tal y como se configure en la aplicación. Cada

lote de entrada conforma un RDD, con lo que se puede transformar o externalizar como salida. Un diagrama de la arquitectura se puede ver en la figura 4.13.

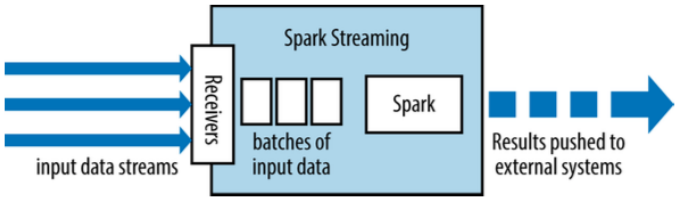


Figura 4.13: Arquitectura de Spark Streaming.

Esta abstracción, como se ha indicado, se denomina *DStream*, y se puede ver en la figura 4.14, donde queda patente que no es más que una secuencia de RDDs, donde cada RDD tiene una porción de los datos en el flujo organizado temporalmente.

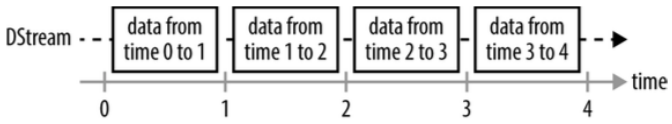


Figura 4.14: Modelo temporal DStream.

Aparte de las operaciones de transformación habituales en los RDDs, los *DStreams* soportan unas transformaciones denominadas *statefull* que pueden agregar datos a través del tiempo. Las operaciones de salida que soportan los *DStreams*, si bien son análogas a las acciones que se aplican a los RDDs, con la salvedad de que son ejecutadas periódicamente a cada paso temporal establecido.

La arquitectura de ejecución de *Spark Streaming* dentro de los componentes *driver-worker* se muestra en la figura 4.15. Para cada fuente de entrada, se lanza un receptor, que no es más que una tarea en ejecución dentro de los ejecutores del clúster, ocupándose de recoger los datos de la fuente y salvando los mismos como RDDs. Estos ejecutores que reciben los datos, automáticamente replican la información a otro ejecutor del clúster para mantener la tolerancia a fallos. Los datos recibidos se almacenan en la memoria de los ejecutores, tal y como ocurre con los RDDs cacheados. El contexto de ejecución se denomina *StreamingContext*, reside en el *driver* y se ocupa de ejecutar periódicamente trabajos Spark para procesar los datos, combinando los resultados con RDDs de otras series temporales.

Spark Streaming ofrece la misma tolerancia a fallos para los *DStreams* que aporta Spark en los RDDs: siempre y cuando exista una copia de los datos de entrada, se puede volver a calcular cualquier estado derivado haciendo uso de la herencia (*lineage*), con la penalización temporal correspondiente de volver a calcular los pasos dados. Por defecto, los datos se replican en dos nodos, por lo que se pueden tolerar fallos de un único ejecutor. También se tiene por defecto un procedimiento que se denomina *checkpointing*, por el que cada 5 - 10 lotes se salva el estado de los datos, con lo que en caso de fallo, al recuperarse de datos perdidos, sólo se necesitará volver desde el último salvado.

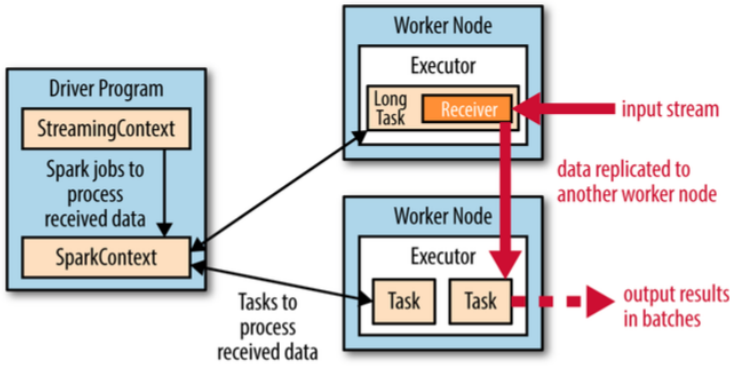


Figura 4.15: Arquitectura driver-executor Spark Streaming.

Las transformaciones sobre DStreams contemplan los siguientes tipos:

- Se denominan "sin estado" ó *Staless* cuando el procesamiento de cada bloque no depende de los datos de los anteriores. Entre ellas se encuentran: *map()*, *flatMap()*, *filter()*, *repartition()*, *reduceByKey()*, *groupByKey()*. Estas transformaciones también pueden agrupar o combinar datos de otros *DStreams*, por ejemplo en mapas de clave/valor se dispone de transformaciones de agrupación: *cogroup()*, *join()*, *leftOuterJoin()*, etc. Las operaciones anteriores se usan para realizar operaciones sobre el conjunto de los datos de forma unitaria en cada bloque. Así mismo se pueden mezclar varios *DStreams* mediante la transformación *union()*. Para concluir, se dispone de otro operador denominado *transform()* que permite proporcionar cualquier función de transformación que se disponga RDD a RDD, permitiendo la reusabilidad del código ya generado para procesos de lotes estándar.

- Se entienden "con estado", *Statefull* cuando se usan datos o resultados

intermedios de bloques anteriores. Estas incluyen transformaciones basadas en ventanas deslizantes y en el seguimiento de estado a lo largo del tiempo, que es usar datos de etapas previas en la etapa actual. Los dos tipos son: ventanas deslizantes (*Sliding Windows*) ó *updateStateByKey()*. El segundo caso es para mantener objetos asociados a cada evento, como puedan ser identificativos de sesión o datos semejantes.

- El primer tipo, o ventanas deslizantes, requieren tener activo el modo *CheckPointing* y se configuran sobre dos parámetros: duración de la ventana y duración del desplazamiento; ambos múltiplos del tiempo "intervalo de lote". La operación que se usa para usar la ventana deslizante es *window(duración ventana, duración desplazamiento)*, que se puede aplicar sobre cualquier *DStream*, ofreciendo operaciones con el resultado como: *reduceByWindow()*, *recudeByKeyAndWindow()*, *countByWindow()*, *countByValueAndWindow()*.
- El segundo tipo se fundamenta en *updateStateByKey()* proporcionando acceso a una variable de estado asociada en los *DStreams* como pares de clave/valor.

Las operaciones de salida que permite *Spark Streaming* pueden ser operaciones de salvado de datos, *save()*, o de escritura en pantalla, *print()*. Dentro de las opciones de salvado, se puede indicar un directorio y que se haga de forma incremental con un parámetro de sufijo opcional, o bien se pueden salvar como un formato de salida típico en Hadoop, *saveAsSequenceFile()*. Así mismo, existe una operación genérica denominada *foreachRDD()*, semejante a *transform()*, que permite acceder a cada RDD dentro del *DStream*, permitiendo reutilizar cualquier función que se disponga de programas de lotes.

Es importante comentar la forma en la que opera *Spark Streaming* con respecto a los receptores y cómo se ejecutan éstos dentro de Spark. Cada receptor es una tarea de larga duración que se ejecuta en un *executor* de Spark, ocupando y reservando CPU para la aplicación. De cara a poner múltiples receptores, se debe tener en cuenta que serán necesarios tantos *core* como número de receptores, además de los necesarios para ejecutar la lógica deseada.

En caso de tener que aumentar el paralelismo del sistema, se trabaja en tres áreas:

- Incremento del número de receptores. Si hay muchos registros que

leer, los propios receptores pueden ser el cuello de botella de la aplicación, por tanto el paso lógico es incrementar el número de éstos. Se pueden añadir más receptores simplemente creando múltiples *DStreams* y aplicando una transformación de *union()* entre ellos.

- Repartionado explícito de la carga. Si los receptores no son el problema, se puede forzar la redistribución de la carga dentro del clúster, aumentando la repartición del *Dstream* correspondiente (acción *repartition*).
- Aumentar el paralelismo en la agregación. En operaciones como *reduceByKey()* se puede especificar el paralelismo deseado como segundo parámetro.

4.11 Arquitectura de ESIDE-BIDS.

La presente tesis trata de establecer un nuevo modelo integral para los Sistemas de Seguridad de la Información, basado en *Big Data*, es por ello que se debe poder integrar en las soluciones existentes de la forma más sencilla posible. El modelo de arquitectura más común que se suele usar en la industria es el de agentes de recogida de información y un servicio centralizador que recupera la información, la analiza y realiza las tareas de notificación oportunas. En algunas arquitecturas complejas, puede haber servicios de agrupamiento por zonas que reporten a otro sistema nivel superior. Todo lo anterior se puede ver en la figura 4.16.

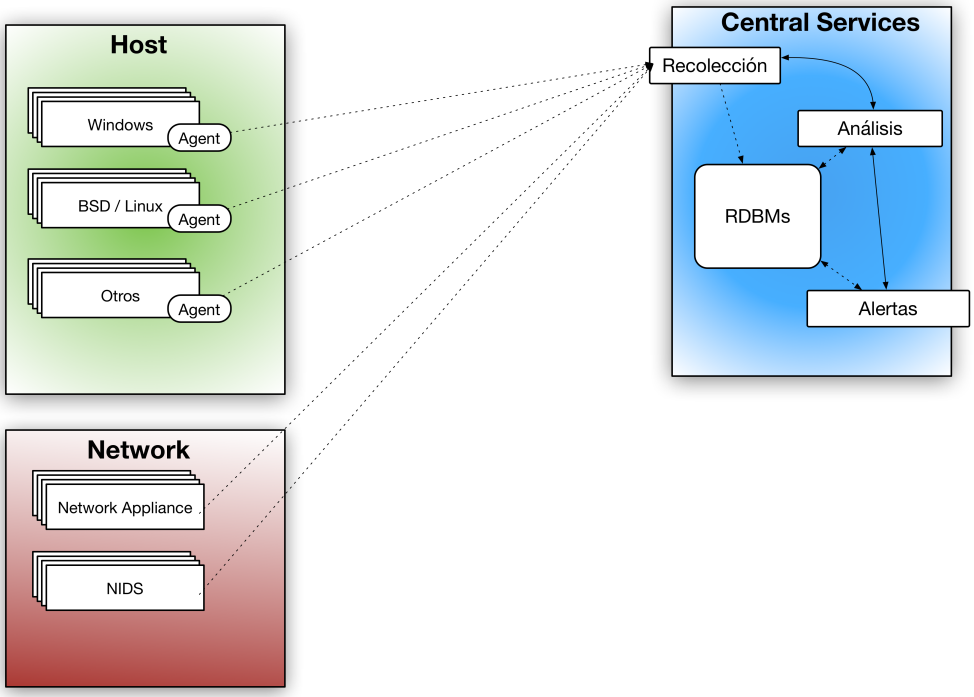


Figura 4.16: Arquitectura General SIEMs.

En general, se tienen varios tipos de recolectores de información, repartidos entre aquellos que requieren de un agente interno y de los que no. Un agente no es más que un servicio o aplicación de software que captura evidencias del sistema operativo en la forma más apropiada, que de forma extendida se basa en la recolección de información de logs del sistema operativo en cuestión. Así pues, aquellos recolectores que no disponen de

agente suelen ofrecer dos fuentes de entrega de conocimiento, los logs que generan en formado crudo que se pasarán a los servicios centrales y aquellos recolectores que ya realicen un procesado de la información antes de subirla a los servicios centrales. En este segundo grupo están los *Network Intrusion Detection Systems*, que desplegados en forma de sistema operativo o bien en forma de *appliance* por la instalación, realizan la captura del tráfico de red y procesan la misma antes de lanzar alertas a la capa de centralización.

Es habitual que los agentes realicen una criba de la información que se envía, teniendo en su interior reglas específicas basadas en conocimiento experto que permiten la selección e identificación de aquellos fragmentos de información relevantes a ser procesados por la capa superior. Tal es así, como se vio en el capítulo de Recogida de evidencias (capítulo 3), que las tecnologías de recogida de información de *host* como OSSEC [Mic15], o las tecnologías de recogida de tráfico de red, como Snort [aiA15], disponen de reglas generalistas en su interior sobre la que comparan la información recabada.

Como primera integración, el modelo de ESIDE-BIDS propone la arquitectura reflejada en la figura 4.17. En ella, se mantienen la infraestructura original, permitiendo al sistema SIEM trabajar por su cuenta como hasta la fecha, pero teniendo un nuevo flujo de información disponible en su base de datos relacional. Esta información es revertida por el modelo ESIDE-BIDs, que como un sensor más, hace una entrega de conocimiento ya procesado al sistema SIEM central, sin pasar por su capa de recolección e insertando directamente los resultados en la base de datos relacional de forma apropiada.

Es importante denotar que en todas las aproximaciones que se exponen a continuación se parte de la base del procesamiento de flujos y del análisis de la información mediante la capa de *machine learning* en cascada que presenta esta tesis.

En la primera aproximación a la solución de *Big Data* aplicada en entornos SIEM, se recoge la información de forma activa, en formato de logs, mediante la tecnología Apache Flume, sección 4.2. Los logs recabados se almacenan en el sistema de archivos distribuido del clúster Hadoop, permitiendo a Spark Streaming analizar la información según van apareciendo ficheros en el directorio de las diferentes fuentes. Tras aplicar la algoritmia necesaria sobre los logs, y correlacionarlos entre ellos de forma adecuada, los resultados se revierten al sistema de archivos HDFS, donde Sqoop (sección 4.3) se ocupará de insertar los resultados finales en un formato explo-

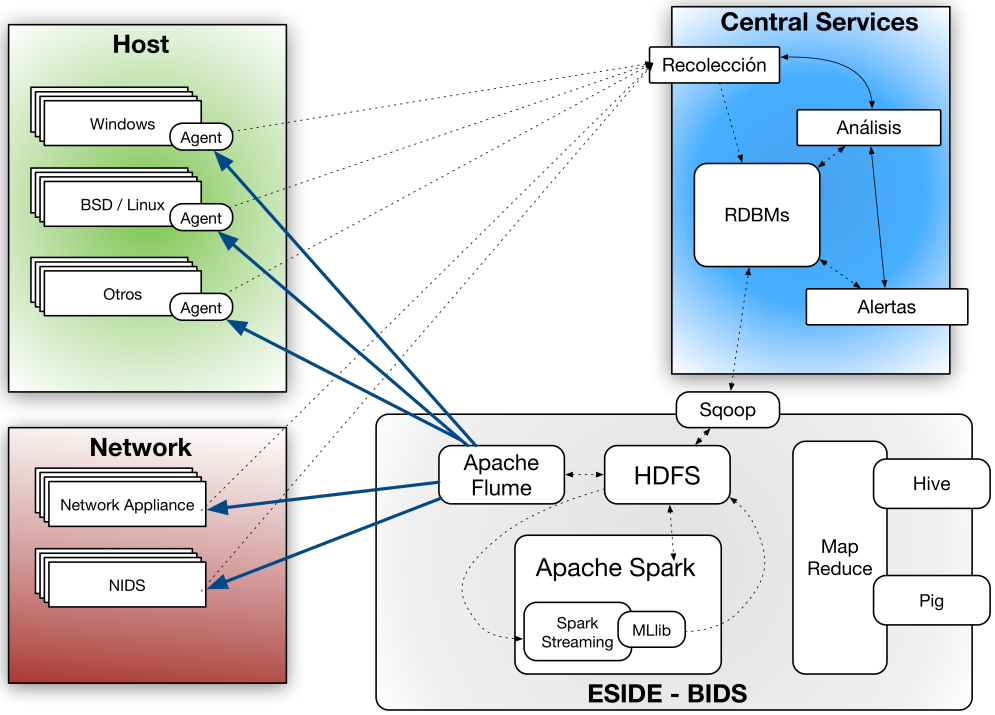


Figura 4.17: Primera integración ESIDE BIDS.

table por el sistema SIEM directamente. Esta primera integración, ofrece además una ventaja competitiva para el análisis de la información, permitiendo a los usuarios finales o administradores del sistema de seguridad, realizar búsquedas detalladas sobre los logs sin filtrar que se recogen, haciendo uso de las herramientas de MapReduce, Hive (sección 4.5) o Pig (sección 4.4).

La siguiente integración evolutiva para la industria, figura 4.18, basándose en el modelo de ESIDE-BIDS, sería directamente evitar que la información se envíe a la capa de recolección del SIEM, ocupándose el entorno de *Big Data* de tratar la información. Para ello se usa una capa de recepción de información que siga las leyes del Big Data: Apache Kafka (sección 4.9). En esta fase, se mantiene la explotación de resultados usando los sistemas de bases de datos relacionales y el resto de desarrollos existentes en el sistema SIEM, pero esta información de los sensores es filtrada por los servicios de *Big Data* mediante trabajos MapReduce o Spark antes de ser almacenada en la base de datos relacional. Este modelo, presenta las siguientes ventajas:

- Se puede escalar la información acorde al número de fuentes sin llegar a saturación de los servicios centrales.
- Se reduce el procesamiento de la información que tienen que filtrar los equipos remotos, ya que todo ocurre en la capa de *Big Data*.
- Se tienen más datos y mayor trazabilidad de los eventos ocurridos, persistidos durante más tiempo.
- Se dispone de herramientas más avanzadas de análisis de datos que las soluciones actuales de SIEMs (basadas en utilidades de Hive y Pig para *Business Intelligence*).

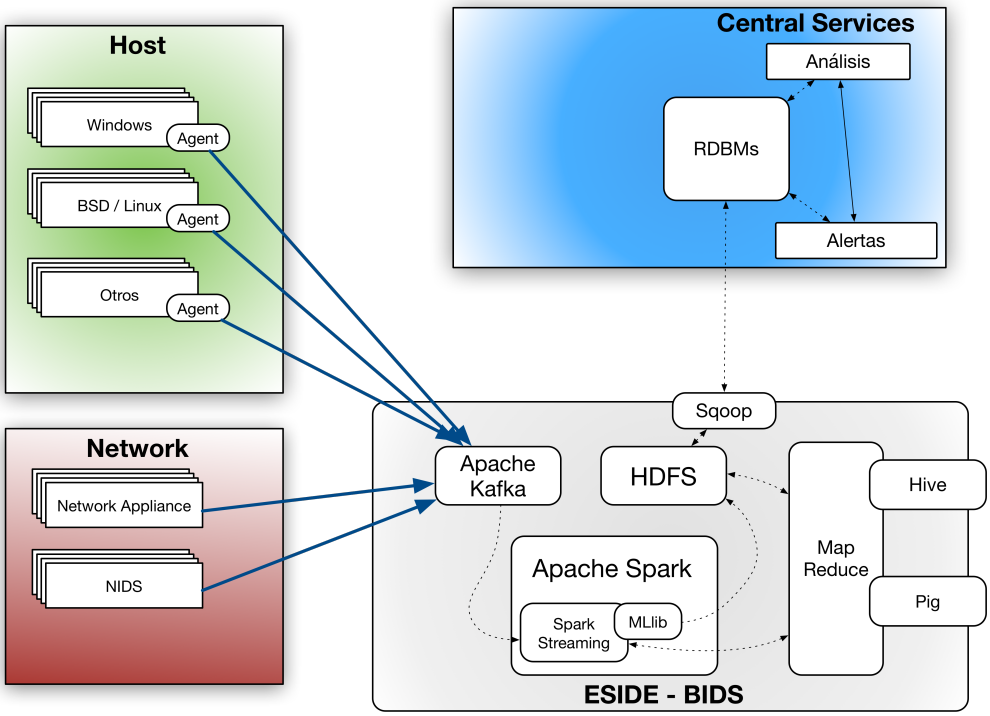


Figura 4.18: Segunda integración ESIDE BIDS.

Finalmente, en la última integración del modelo ESIDE-BIDS, ya se eliminan completamente las bases de datos relacionales y los sistemas de filtrado, análisis y alertas se pasan a trabajos bien MapReduce o bien Spark.

Esta última integración se puede ver en la figura 4.19. La capa de persistencia de esta solución, puede hacer uso de funcionalidades más avanzadas para el almacenamiento de la información, usando HBASE (sección 4.8) [1]

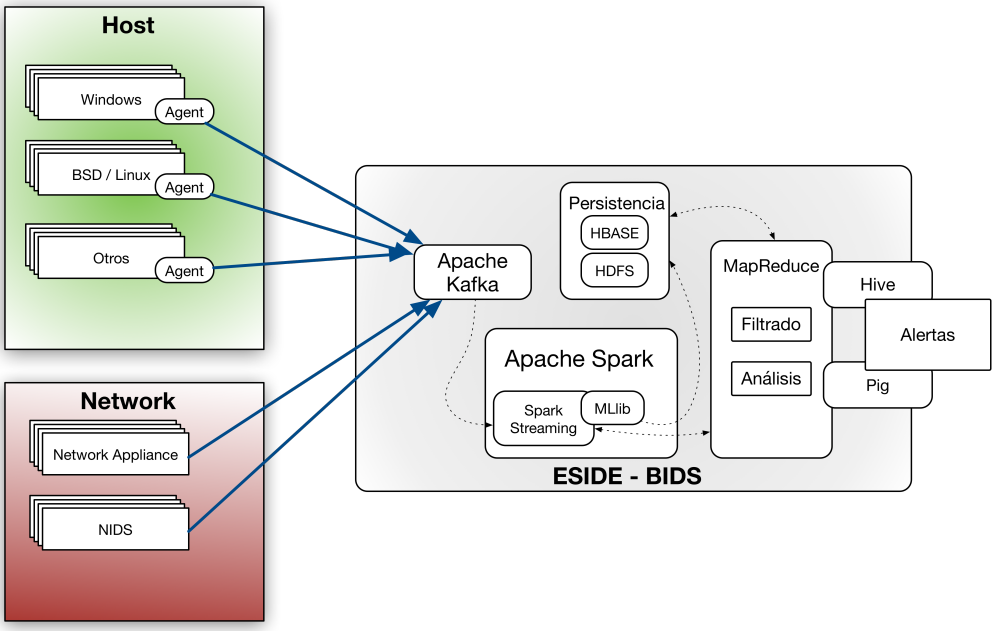


Figura 4.19: Tercera integración ESIDE BIDS.

La solución en la que se demuestra y experimenta la presente disertación es la segunda, si bien se han mantenido las entregas de información de los agentes y la capa de recolección de los servicios centrales, duplicando en muchos casos el tráfico generado. Se irá ahondando en la explicación de cada uno de los componentes y la forma en la que se trabaja, al igual que en los correspondientes experimentos, en los siguientes capítulos.

[1] Las otras dos integraciones también podrían explotar la información de esta forma, aunque no se ha estimado necesario por el grado de madurez en *Big Data* necesario para ello.

Parte II

ESIDE-BIDS. Experimentación y análisis de resultados.

«HAL-9000: - Lo siento, Dave, no puedo hacer eso.»

Arthur C. Clarke (1968)

Capítulo 5

KDDCup99.

En el presente capítulo se hará una introducción de las técnicas y algoritmos de aprendizaje supervisado y no supervisado de las que dispone Apache Spark. Se mostrará tanto el entorno de laboratorio que se dispone para realizar las pruebas localmente, tanto la forma de escalar estos algoritmos a Amazon EC2 [Ama15a], pudiendo disponer de clústeres potentes cuando se requiera el uso de los mismos. En este capítulo las tecnologías principales sobre las que se trabaja son Apache Spark [Fou15a] usando el lenguaje Scala [Sca15].

5.1 Análisis del DataSet.

La denominada KDD Cup es una competición que se organiza por un grupo de interés por parte de ACM. Cada año, se publica un problema de aprendizaje, así como el DataSet asociado, donde los investigadores pueden ofrecer soluciones, así como publicaciones científicas asociadas a su investigación con la mejor solución al problema planteado. En 1999 se publicó un DataSet relacionado con la Detección de Intrusiones.

El fichero que publicaron ya tenía los datos en ya filtrados, ofreciendo vectores de evidencias sobre información ya procesada. El fichero de datos es de 708 MB y contiene 4.9M conexiones, que si bien es grande, no lo es tanto como para hablar de *Big Data*, pero servirá como punto de partida al trabajo que se realiza en la presente tesis. Los datos están formateados en formato CSV, conteniendo 38 características que son:

- Tipo de ataque (23 categorías): *back*, *buffer_overflow*, *ftp_write*, *guess_passwd*, *imap*, *ipsweep*, *land*, *loadmodule*, *multihop*, *neptune*, *nmap*, *normal*, *perl*, *phf*, *pod*, *portsweep*, *rootkit*, *satan*, *smurf*, *spy*,*teardrop*, *warezclient*, *warezmaster*.
- Duración.
- Tipo de protocolo: tcp, udp, etc.
- Servicio de red destino: http, telnet, etc.
- Estado de la conexión normal o erróneo.
- Número de bytes desde origen.
- Número de bytes desde destino.
- Land: 1 si la conexión es del/al mismo host-puerto, 0 en caso contrario.
- Número de fragmentos erróneos.
- Número de paquetes urgentes.
- Número de indicadores "hot".
- Número de logins fallidos.
- Logged_in: 1 si se ha conseguido acceder, 0 en caso contrario.
- Número de condiciones comprometidas.
- Root_shell: 1 si se ha conseguido un shell de root, 0 en caso contrario.
- Su_attemps: 1 si se ha tratado de realizar una elevación de privilegios, 0 en caso contrario.
- num_root: Número de accesos como root.
- Número de ficheros creados.
- Número de shells abiertas.
- Número de ficheros críticos accedidos.
- Número de comandos fuera de límites (por no tener privilegios).

- is_hot_login: 1 si el login es del grupo de usuarios "hot", 0 en caso contrario.
- is_guest_login: 1 si el login es de usuarios invitados, 0 en caso contrario.
- Número de conexiones al mismo host como la conexión actual en los últimos dos segundos (mismo hosts).
- Mismo host: Porcentaje de conexiones que tienen errores SYN.
- Mismo host: Porcentaje de conexiones que tienen errores de rechazo (REJ).
- Mismo host: Porcentaje de conexiones al mismo servicio.
- Mismo host: Porcentaje de conexiones a servicios diferentes.
- Número de conexiones al mismo servicio que la conexión actual en los últimos 2 segundos (mismo servicio).
- Mismo servicio: Porcentaje de conexiones que tienen errores SYN.
- Mismo servicio: Porcentaje de conexiones que tienen errores de rechazo (REJ).
- Mismo servicio: Porcentaje de conexiones a otros hosts.

Si bien muchas de las características del dataset son categóricas, en los casos en los que están son 0 y 1, resulta funcionar bastante bien aplicar técnicas numéricas pese a no existir comparación entre una categoría y otra como en las características numéricas. Inicialmente vamos a usar la shell de Spark en lenguaje Scala para evaluar el fichero de datos, código 5.1. Se puede apreciar en un primer vistazo que hay más evidencias clasificadas como ataque que como tráfico normal, lo que resultaría extraño para una situación real.

Código 5.1: Conteo de total de casos por categoría

```
% klobato@ubuntu1:/usr/local/hadoop-2.6.0/spark-1.4.1-bin-hadoop2.6$ bin/spark-shell
    --master "spark://ubuntu1:7077" --num-executors 3 --driver-memory 16g
    --executor-memory 8g --executor-cores 6
scala> val rawData = sc.textFile("hdfs://ubuntu1:9000/user/klobato/kddcup.data")
scala> rawData.map(_.split(',').last).
countByValue().toSeq.sortBy(_._2).reverse.foreach(println)
```

```
(smurf.,2807886)
(neptune.,1072017)
(normal.,972781)
(satan.,15892)
(ipsweep.,12481)
(portsweep.,10413)
(nmap.,2316)
(back.,2203)
(warezclient.,1020)
(teardrop.,979)
(pod.,264)
(guess_passwd.,53)
(buffer_overflow.,30)
(land.,21)
(warezmaster.,20)
(imap.,12)
(rootkit.,10)
(loadmodule.,9)
(ftp_write.,8)
(multihop.,7)
(phf.,4)
(perl.,3)
(spy.,2)
```

En total se tienen 23 etiquetas, distribuidas acorde a histograma de la figura 5.1, que se pueden entender como categorías diferentes, dentro del conjunto de datos de KDDCup99, de las cuales los ataques *smurf* y *neptune* son los más frecuentes. Así mismo, hay varias características que son categóricas y no numéricas.

5.1.1 Preparación de los datos.

El siguiente paso a realizar es la lectura de los datos, procediendo a la creación de los correspondientes vectores de evidencias, y dado que hay evidencias categóricas se van a cambiar por medio de la técnica *one-hot encoding*, que simplemente traslada cada categoría a una evidencia nueva. Como ejemplo de la técnica anterior, para "protocolo" se tienen tres valores: tcp, udp e icmp, que se desarrollarán como tres nuevas características: es_TCP, es_UDP, es_ICMP; siendo TCP traducido por [..,1,0,0,...], ICMP por [...,0,0,1,...] y UDP por [...,0,1,0,...] . La lectura de las evidencias y su adaptación a vectores se puede ver en el código 5.2.

Código 5.2: Creación de vectores con one-hot encoding para categorías.

```
def buildDataSet(rawData: RDD[String]): (String => (String,Vector)) = {
      val splitData = rawData.map(_.split(','))
      val protocols = splitData.map(_(1)).distinct().collect().zipWithIndex.toMap
      val services = splitData.map(_(2)).distinct().collect().zipWithIndex.toMap
      val tcpStates = splitData.map(_(3)).distinct().collect().zipWithIndex.toMap
```

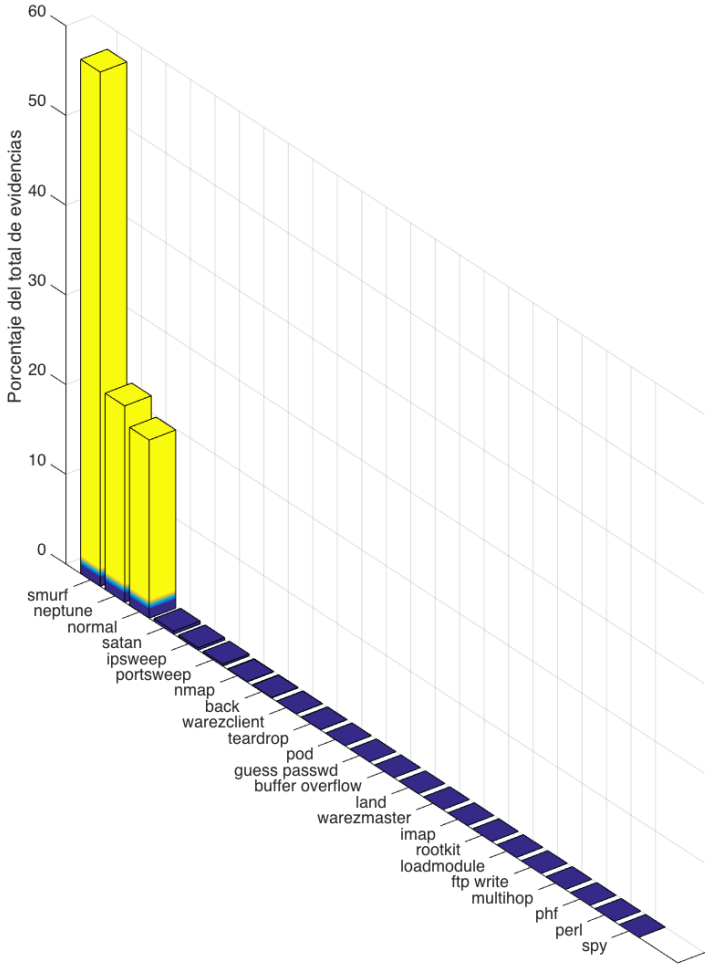


Figura 5.1: Apache Kafka - Componentes.

```
(line: String) => {
 val buffer = line.split(',').toBuffer
 val protocol = buffer.remove(1)
 val service = buffer.remove(1)
 val tcpState = buffer.remove(1)
 val label = buffer.remove(buffer.length -1)
 val vector = buffer.map(_.toDouble)

 val newProtocolFeatures = new Array[Double](protocols.size)
 newProtocolFeatures(protocols(protocol)) = 1.0
 val newServicesFeatures = new Array[Double](services.size)
 newServicesFeatures(services(service)) = 1.0
 val newTcpStateFeatures = new Array[Double](tcpStates.size)
 newTcpStateFeatures(tcpStates(tcpState)) = 1.0
```

```
        vector.insertAll(1, newTcpStateFeatures)
        vector.insertAll(1, newServicesFeatures)
        vector.insertAll(1, newProtocolFeatures)
        (label, Vectors.dense(vector.toArray))
       }
}
```

La normalización de las caraterísticas, ecuación 5.1, es un paso importante a la hora de estandarizar los vectores de evidencias. Para ello se debe restar a cada valor de las características la media y dividir todo por la desviación típica, tal y como se muestra en la siguiente ecuación:

$$n_i = \frac{f_i - \mu_i}{\sigma_i} \tag{5.1}$$

Dónde la media μ_i y la desviación estándar σ_i viene definidas por las ecuaciones 5.2 y 5.3 respectivamente.

$$\mu_i = \frac{1}{N}\sum_{i=1}^{N} f_i \tag{5.2}$$

$$\sigma_i = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(f_i - \mu_i)^2} \tag{5.3}$$

La implementación en el lenguaje Scala usando programación funcional se puede ver en el código 5.3. Se debe tener en cuenta que para poder procesar en paralelo, la forma en la que se programa en los lenguajes funcionales es diferente a la programación tradicional.

Código 5.3: Normalización de evidencias.

```
def buildNormalizedValues(data: RDD[Vector]) : ( Vector => Vector) = {
       val dataAsArray = data.map(_.toArray)
       val numCols = dataAsArray.first().length
       val n = dataAsArray.count()
       val sums = dataAsArray.reduce((a, b) => a.zip(b).map(t => t._1 + t._2))
       val means = sums.map(_ / n)

       val sumSquares = dataAsArray.aggregate(new Array[Double](numCols))(
       (a, b) => a.zip(b).map(t => t._1 + t._2 * t._2),
       (a, b) => a.zip(b).map(t => t._1 + t._2)
       )

       val stdevs = sumSquares.zip(sums).map {
       case(sumSq, sum) => math.sqrt(n*sumSq - sum*sum) / n
       }

       (datum: Vector) => {
```

```
            val normalizedArray = (datum.toArray, means, stdevs).zipped.map{
            (value, mean, stdev) => if(stdev <= 0) (value - mean) else (value - mean) /
                 stdev
            }
            Vectors.dense(normalizedArray)
        }
}
```

5.2 Clusterización mediante K-means.

El principal problema que se aborda en la detección de intrusiones son las anomalías, es decir, encontrar aquellos eventos que se salgan de lo habitual y establecido como normal. En el caso de disponer de una base de datos suficientemente grande con la clasificación de los mismos, se podría pensar en partir de una solución de clasificación, sobre la que poder aplicar las técnicas de aprendizaje supervisado que existen en la literatura.

Los algoritmos de clústerización son los más conocidos a la hora de realizar un aprendizaje no supervisado, tratando de encontrar grupos naturales en los datos. Una técnica efectiva es la denominada *K-means*, en la que se trata de detectar k clústeres dentro de un conjunto de datos. El número de grupos es un valor a establecer por el *data scientist*, siendo k un hiperparámetro del modelo y dependiente completamente del *dataset*. Generalmente para medir la distancia que separa unas evidencias de otras se usa la distancia Euclídea, definida para datos de tipo numérico. Dentro del algoritmo de *K-means*, un grupo es identificado por su centroide, que es la media aritmética de todos los puntos pertenecientes al clúster en cuestión. Para empezar el algoritmo coge algunos puntos como centroides de los grupos y asigna el resto al centroide más cercano, después por cada clúster, se computa un nuevo centroide con la media de los puntos asociados al mismo, repitiendo el proceso hasta encontrar la mejor solución al corpus de datos.

Volviendo al tema de la detección de intrusiones, se sabe que algunos exploits siguen patrones conocidos, que se pueden identificar apriori: escaneo de puertos como primer paso de un ataque remoto. Computando el número de puertos accedidos desde un host remoto en un corto periodo de tiempo, puede obtenerse un indicador bastante bueno para detectar este tipo de ataques. Lo mismo puede establecerse con otros ataques teniendo en cuenta el número de bytes enviados y/o recibidos, errores TCP, etc.

La pega viene cuando se deben detectar ataques no conocidos, o denominados *zero-days*, y para ello se aborda la investigación en sistema de detección de anomalías. Puede que sólo detecten situaciones extrañas que no

sean un ataque en sí mismos, pero desde luego merece la pena analizarlas. En este campo, muchas investigaciones se hace uso del mencionado algoritmo de *K-means*, por citar algunas de las más actuales: [MYSU14], [Beg14], [SJ14], [GW14], [MJ+14], [EV14], etcétera.

En general las técnicas de clústering se aplican cuando no se dispone de una clase asignada, pero las instancias o evidencias pueden ser divididas de forma natural en grupos. La matemática asociada puede verse en las ecuaciones 5.4 y 5.5, pero básicamente la técnica K-means consiste en lo siguiente:

1. Se especifica de antemano el número de clústeres se buscan. Es número de grupos es el número K del algoritmo.

2. Se escogen K puntos de todo el dataset de forma aleatoria como inicialización del algoritmo.

3. Se asignan todas las instancias al clúster más cercano en función de la distancia (en general Euclídea).

4. Se calcula el centroide, o la media, de todas las instancias en cada clúster (esta es la parte de *means*).

5. Se toma estos centroides como el nuevo centro de sus respectivos grupos.

6. Se repite el proceso iterativamente hasta que salen los mismos centroides y se asignan los mismos puntos a los mismos grupos.

Dado un conjunto inicial de clústeres $m_1,...m_k$ y usando la Distancia Euclídea:

- Etapa de asignación: asignar todos los puntos al centroide más cercano en tiempo (t).

$$S_i^{(t)} = \|x_p - m_i^{(t)}\|^2 \leq \|x_p - m_j^{(t)}\|^2 \quad \forall j, 1 \leq j \leq k \tag{5.4}$$

- Etapa de actualización: calcular las medias del clúster para establecer los nuevos centroides en $(t+1)$.

$$m_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j \tag{5.5}$$

El método de clústerización de K-means es muy sencillo y efectivo, ya que se basa en la elección de aquellos centros de grupo como centroides, que minimizan la distancia de cada punto del clúster con su centroide. Una vez que la iteración ha estabilizado, cada punto es asignado al clúster más cercano, por lo que el efecto es el de minimizar la distancia cuadrada total de todos los puntos, aunque esto es un mínimo local: es decir, no hay garantía de que sea el mínimo global. Esto quiere decir que los grupos finales son muy dependientes de la elección aleatoria inicial, consiguiendo resultados muy diferentes dependiendo de los puntos iniciales. Se debe denotar que: es prácticamente siempre imposible encontrar globalmente los clústeres óptimos, así que normalmente lo que se suele hacer es correr el algoritmo varias veces y seleccionar aquel que tenga mejores resultados, o dicho de otra manera, aquel que tenga la distancia cuadrada total más pequeña.

La variante conocida como **k-means++** se ocupa de elegir mejor los centroides iniciales, conocidos generalmente como *seeds*. En lugar de empezar con unos *seeds* aleatorios, se escoge el primer *seed* aleatoriamente sobre una distribución uniforme de probabilidad en el espacio completo del *dataset*. Después se escoge el segundo *seed* con una probabilidad proporcional al cuadrado de la distancia con el primero. A cada etapa se escoge el siguiente *seed* con una probabilidad proporcional al cuadrado de la distancia con el *seed* más cercano. Se consigue mejorar notablemente los resultados y es la implementación que está hecha en Apache Spark MLlib [Fou15b].

5.2.1 **Búsqueda del número de grupos en el clúster: K.**

El siguiente objetivo es la definición del número K correcto, que como mínimo debería ser de 23 que son las etiquetas que tiene ya asignadas el *DataSet*. Para poder comparar lo bueno que es un modelo con otro, se va a hacer uso de dos características:

- La distancia euclídea de los puntos del clúster con el centroide
- La entropía que tiene cada clúster con respecto a las etiquetas que tiene asignadas.

Contra mejor sea la distancia media de los puntos con el centroide, mejor será la cercanía del grupo con mismo, facilitando la clusterización, para ello se hará uso de las funciones expresadas en el código 5.4.

Código 5.4: Distancia de los puntos con los centroides.

```
def distance(a: Vector, b: Vector) =
      math.sqrt(a.toArray.zip(b.toArray).map(p => p._1 - p._2).map(d => d * d).sum)

def distToCentroid(datum: Vector, model: KMeansModel) = {
      val cluster = model.predict(datum)
      val centroid = model.clusterCenters(cluster)
      distance(centroid, datum)
}

def clusteringScore(data: RDD[(String, Vector)], model: KMeansModel): Double = {
      data.map{ case(s, d) => distToCentroid(d, model) }.mean()
}
```

El cálculo de la distancia expresado en el código 5.4 se puede leer de atrás adelante como sigue: el sumatorio de los cuadrados de las diferencias de los vectores en cada uno de los elementos de los vectores A y B, calculando finalmente la raíz cuadrada del mismo.

Obviamente, si se igualase K al número de datos distintos que se tienen, la distancia media sería de 0, aunque en este caso no se hablaría de agrupación de datos. Paradójicamente a la hora de buscar el K perfecto, puede ocurrir que pese a usar un K mayor se empeore la distancia, lo que no debería producirse, dado que a mayor K se debería obtener como mínimo los mismos resultados que en el anterior con K menor. Esto ocurre debido a que el algoritmo no siempre es capaz de encontrar el *clustering* óptimo para un valor de K, ya que siempre se parte de una inicialización aleatoria que consiga un mínimo local, que es suficiente pero no óptimo. Las implementaciones de K-means|| y K-means++ son variaciones del algoritmo K-means, de forma que sea paralelizable y más robusto [NC15]. Para controlar esta situación, se hace uso de dos parámetros: ϵ y el número de veces a calcular un modelo partiendo de la inicialización aleatoria (*setRuns*). Cambiando el threshold ϵ se controla la cantidad mínima que debe desplazarse el centroide para ser considerado un cambio significativo, a valores más bajos el centroide continuará moviéndose durante más tiempo.

La segunda medida que se usará para evaluar la bondad de la K elegida es la entropía que hay en cada clúster teniendo en cuenta las etiquetas que están asignadas en el *DataSet* de KDDCup99. La definición de entropía viene dada por la Teoría de la Información y calcula el nivel de incertidumbre que tiene una colección de valores objetivo en el subconjunto. Se define por la siguiente ecuación 5.14:

$$I_E(p) = \sum_{i=1}^{N} p_i log\left(\frac{1}{p}\right) = -\sum_{i=1}^{N} p_i log(p_i) \tag{5.6}$$

Para poder establecer en K más apropiado, se ha lanzado una búsqueda en el clúster de máquinas de Amazon EC2 haciendo uso del código 5.5. El experimento consiste en la carga del fichero de datos, la normalización del mismo, y la creación de modelos con los parámetros siguientes:

- Número de grupos desde 10 a 200, tomados de 20 en 20.
- Prefijado $\epsilon = 10^{-}6$.
- Buscar 10 modelos para K objetivo, haciendo que se maximice el modelo buscado.
- Imprimir resultados en formato CSV: k, distancia media de las evidencias con los centroides, entropía de cada grupo, tiempo de ejecución.

Código 5.5: Cálculo de los modelos K-means.

```
def main(args: Array[String]) : Unit = {

      if(args.length != 5)
            println("ClusterKDDCup: Introduzca los argumentos requeridos: in_file kMin
                kMax kInc out_file")
      else {
            val sc = new SparkContext(new SparkConf().setAppName("kmeansKDDCup"))

            val rawData = sc.textFile(args(0))

            val data = rawData.map(buildDataSet(rawData))
            val normalizeddata = data.mapValues(buildNormalizedValues(data.values)).cache()

            val minK = args(1).toInt
            val maxK = args(2).toInt
            val incK = args(3).toInt
            val results: IndexedSeq[String] =
                  (minK to maxK by incK).map{ k =>
                        val t0 = System.nanoTime()
                        val kmeans = new KMeans()
                        kmeans.setK(k)
                        kmeans.setRuns(10)
                        kmeans.setEpsilon(1.0e-6)
                        val model = kmeans.run(normalizeddata.values)
                        val t1 = System.nanoTime()
                        "" + k + "," + clusteringScore(normalizeddata, model) +","
                            +clusterScoreWithEntropy(normalizeddata,model)+ ","
                            +(t1-t0)+"ns"
            }
            sc.parallelize(results.toSeq).saveAsTextFile(args(4).toString())
            normalizeddata.unpersist()
      }
}
```

El tiempo total de ejecución ha sido de 2,3 horas, tal y como se puede apreciar en la figura 5.2. Sobre estos modelos, de lo que se trata es de buscar la mejor combinación de parámetros del modelo, de forma que represente lo mejor posible a los datos en estudio. Los resultados del código anterior devuelven los datos necesarios para la visualización de la evolución de la Distancia de las evidencias a los centroides, así como la evolución de la Entropía, que se pueden ver en la figura 5.3. Así mismo, la gráfica del tiempo que se tarda en calcular cada uno de los modelos con el clúster que se ha creado para este experimento aparece en la figura 5.4.

Completed Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20150904033040-0000	kmeansKDDCup	32	12.6 GB	2015/09/04 03:30:40	root	FINISHED	2,3 h

Figura 5.2: Amazon EC2, Tiempo cálculo modelos K-Means.

Como ya se había dicho, es posible que aunque el algoritmo encuentre una solución que cumpla los requisitos impuestos para el clúster K buscado, no sea la solución óptima al problema. Es por ello que se pueden dar casos contradictorios, como sucede con la Entropía en el modelo $K = 70$, o con la distancia en los modelos $K = 150$ y $K = 170$ que prácticamente no mejora. Viendo la evolución de los parámetros, nos podemos quedar con un $K = 150$, ya que subir un poco más no aporta gran valor y un incremento mucho mayor podría dar lugar a un *overfitting*. Otro dato muy representativo y que se debe tener en cuenta en modelos reales, es el tiempo de computación que requiere el cálculo del modelo completo, que como se puede apreciar en la figura 5.4, entre el modelo $K = 150$ y el modelo $K = 170$ son 12 minutos adicionales de cálculo. Los datos en crudo de este experimento se encuentran en la tabla 5.1:

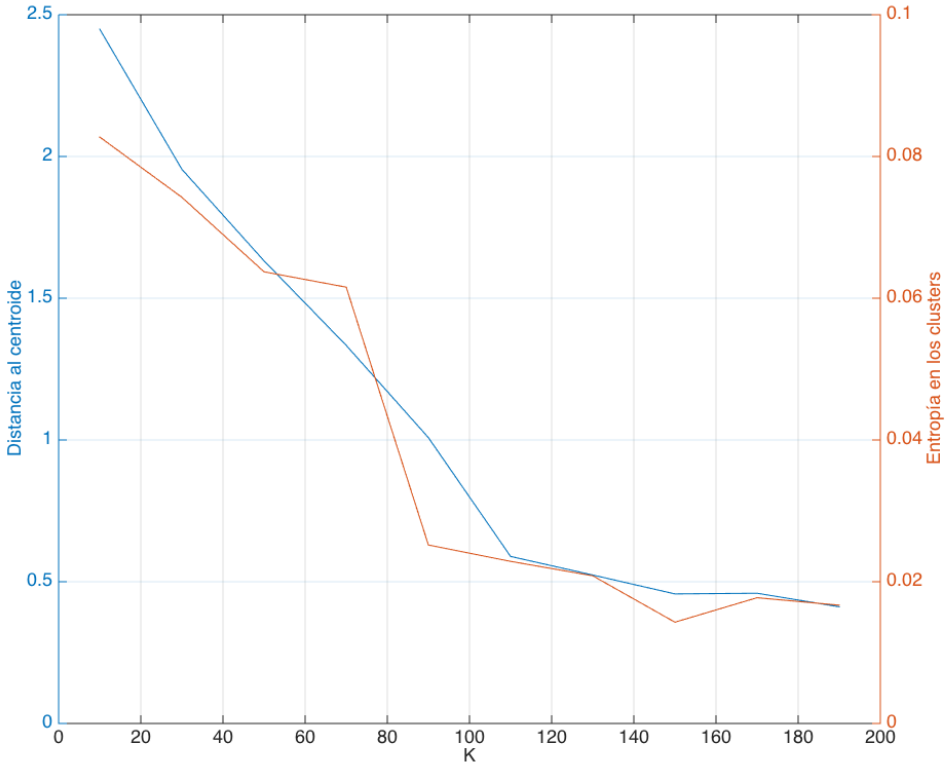


Figura 5.3: Resultados búsqueda de K óptimo.

Cuadro 5.1: Datos crudos cálculo K-Means óptimo.

K	Distancia Centroide	Entropía Etiquetas	Tiempo - Minutos
10	2,449350291	0,08274293	26,64
30	1,954760156	0,074240995	21,28
50	1,631724755	0,06372892	26,00
70	1,333632057	0,061560851	32,38
90	1,008270977	0,025181567	31,79
110	0,589258643	0,022877014	27,17
130	0,523666709	0,020836443	30,11
150	0,45704183	0,014291701	37,65
170	0,45938618	0,017744357	45,24
190	0,411557439	0,016688182	40,96

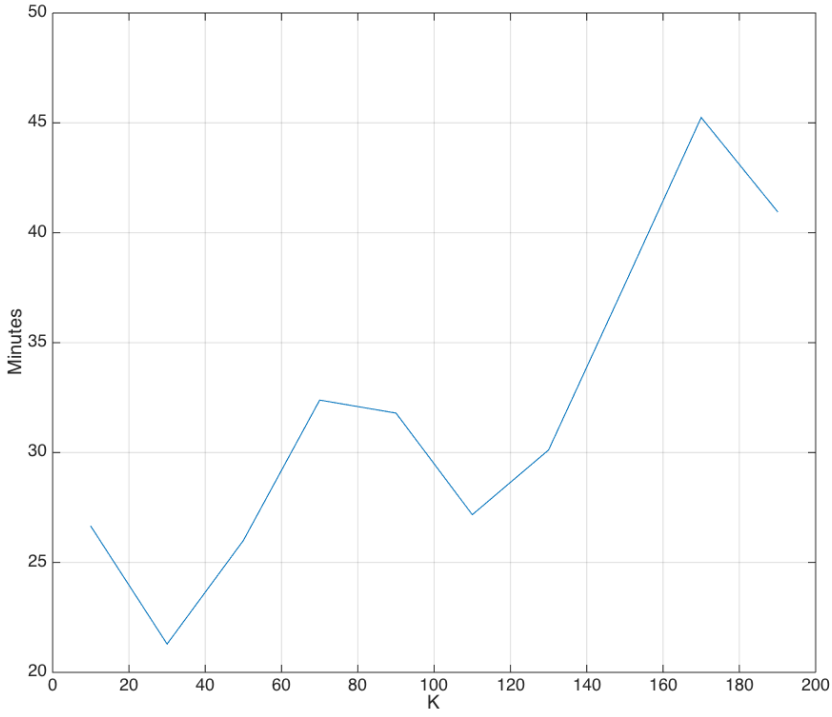


Figura 5.4: Tiempo empleado en el cálculo de los modelos de K-means.

5.2.2 Explotación de resultados: Detección de anomalías.

Para explotar los resultados, se va a proceder a examinar el modelo creado y así poder hacerse una idea de cómo de representativo es sobre el *DataSet* original. El código 5.6 muestra tanto el lanzamiento del modelo con los parámetros seleccionados en el apartado anterior. Así mismo, se pasan todos los datos de entrada por el modelo, calculando el clúster al que pertenece cada evidencia, y la distancia con el centroide del mismo.

Código 5.6: Explotación de Resultados

```
    ...
    val kmeans = new KMeans()
    kmeans.setK(150)
    kmeans.setEpsilon(1.0e-6)
    kmeans.setRuns(10)
    kmeans.run(normalizeddata.values)

    //Obtenemos para cada dato en una tripleta: cluter, etiqueta, distancia al centroide
val clusterLabelDist = normalizeddata.map{ case (label, datum) =>
      val cluster = model.predict(datum)
      val dist = distToCentroid(datum, model)
      (cluster, label, dist)
}.cache()

    //Calculo de la distancia media y la stddev de los puntos en los clusteres
    val clusterDistMedia = clusterLabelDist.map{case (c,l,d) => (c,d)}.groupByKey().map{
        case (c, dlst) =>
        (c, dlst.map(d => (d,1)))
        }.map { case (cluster, distCounter) =>
        val s = distCounter.foldLeft((0.0, 0.0, 0)) {
        (rTriple, caso) => (rTriple._1 + caso._1, rTriple._2 + (caso._1 * caso._1),
            rTriple._3 + caso._2)
        }
        (cluster, s._1, s._2, s._3)
        }.map{ case (cluster, sum, sumSq, n) =>
        val mean = sum/n
        val stdev = math.sqrt(n*sumSq - sum*sum) / n
        (cluster, mean, stdev)
    }.collect().foreach(println)

    clusterLabelDist.map{case(c,l,d) => (l,c)}.groupByKey().map{case (l, clusteres) =>
          val clusteresReduce = clusteres.groupBy(w => w)
          val cCount = clusteresReduce.map(c => (c._1,
              c._2.size)).toSeq.sortBy(_._2).toMap
          (l, cCount)
    }.sortBy(_._1).collect().foreach(println)
    ...
```

El cálculo anterior se cachea de forma que se pueda explotar los datos en memoria de forma más rápida. El siguiente paso que se muestra en el código 5.6 es el cálculo de las distancias medias por clúster y la desviación estándar que se tiene en cada uno. Los resultados se pueden ver en la figura

5.5.

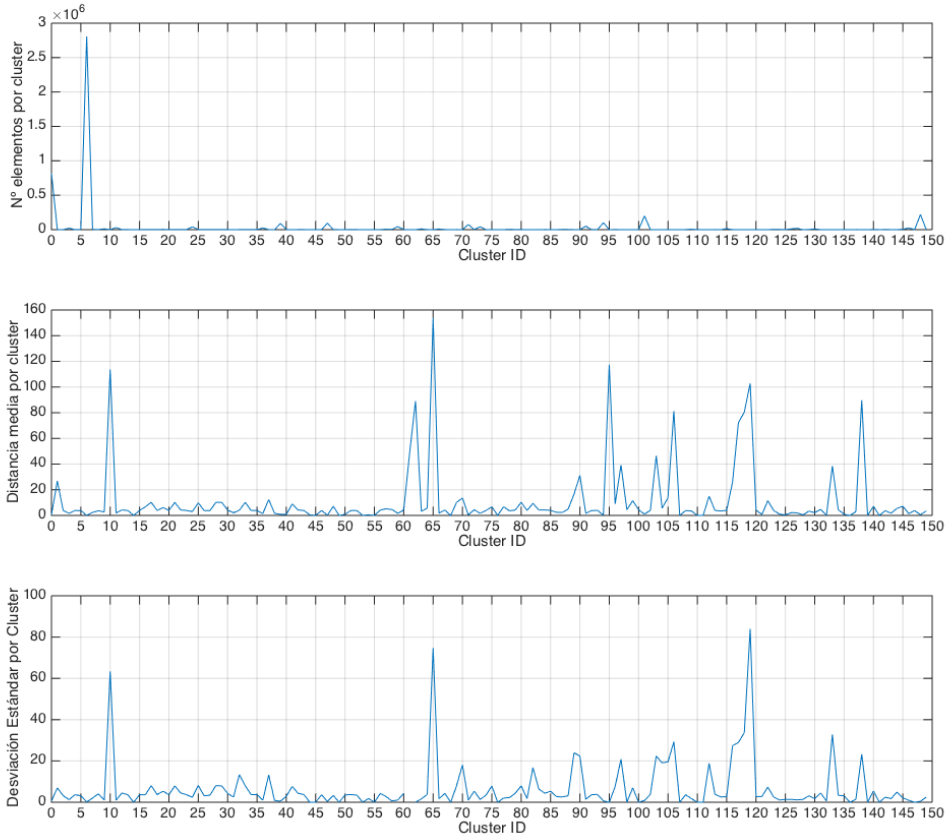


Figura 5.5: Número de elementos, distancia media y desviación por clúster.

Cuadro 5.2: Etiquetas en relación a los clústeres (clúster ->Número de evidencias).

Etiqueta	Cluster ->Contador Casos
back	148->1216, 36->381, 39->280, 101->181, 142->91, 11->43, 75->5, 145->3, 37->2, 127->1
buffer overflow	105->18, 24->6, 19->4, 135->1, 85->1
ftp write	24->4, 129->2, 117->1, 65->1
guess passwd	1->49, 147->2, 10->1, 19->1
imap	18->7, 149->4, 37->1
ipsweep	66->8173, 140->3075, 123->819, 94->162, 124->40, 25->15, 79->13, 30->13, 72->13, 67->13, 12->13, 22->13, 96->13, 60->13, 84->13, 57->13, 24->13, 42->13, 56->13, 91->11, 85->9, 19->4, 127->3, 90->1
land	99->21
loadmodule	19->3, 104->2, 24->2, 135->1, 105->1
multihop	24->3, 129->2, 105->1, 104->1
neptune	0->822501, 94->99768, 47->93222, 25->2000, 24->1798, 19->1723, 57->1600, 108->1066, 12->1046, 20->1044, 42->1044, 18->1043, 92->1043, 33->1043, 69->1043, 43->1042, 84->1042, 93->1042, 67->1041, 35->1041, 13->1041, 17->1040, 96->1040, 28->1040, 74->1040, 23->1039, 72->1039, 22->1039, 34->1039, 15->1038, 51->1038, 30->1038, 113->1038, 102->1038, 60->1038, 4->1037, 48->1037, 56->1037, 79->1036, 5->1036, 29->1035, 46->1035, 21->1034, 2->1033, 52->1032, 26->1030, 27->1030, 80->1027, 8->1011, 16->846, 135->842, 50->837, 40->834, 121->832, 145->488, 54->200, 128->200, 85->200, 132->200, 125->200, 91->93, 73->16, 11->1, 63->1
nmap	32->1030, 66->990, 71->250, 123->36, 124->6, 136->1, 8->1, 139->1, 149->1
normal	148->217436, 101->199780, 39->89292, 71->72925, 91->49986, 59->44503, 73->41114, 24->37445, 11->30282, 146->24564, 36->24518, 3->22270, 127->19627, 9->12577, 126->11543, 63->11027, 130->10885, 109->6828, 78->5375, 57->4998, 145->4741, 7->3833, 123->3693, 87->3534, 124->3410, 88->3056, 48->2328, 129->2123, 19->1816, 64->1103, 16->910, 144->538, 96->503, 86->496, 25->491, 37->486, 104->350, 143->328, 41->299, 105->216, 122->194, 131->181, 75->153, 31->148, 82->129, 90->95, 66->72, 140->62, 85->50, 1->48, 103->46, 112->45, 6->44, 106->41, 12->38, 119->25, 114->25, 116->24, 70->13, 138->11, 53->9, 142->8, 47->7, 99->7, 72->7, 118->7, 38->7, 133->7, 10->7, 97->6, 135->4, 65->4, 117->4, 83->3, 40->3, 18->3, 45->3, 32->2, 61->2, 68->1, 94->1, 107->1, 76->1, 54->1, 134->1, 132->1, 14->1
perl	105->3
phf	105->4
pod	77->259, 38->5
portsweep	58->4461, 142->2067, 115->1642, 94->345, 120->292, 134->285, 83->263, 47->191, 0->142, 89->117, 145->62, 70->44, 60->15, 30->14, 43->14, 42->14, 79->13, 67->13, 113->13, 17->13, 28->13, 102->13, 84->13, 25->13, 22->12, 56->12, 72->11, 50->11, 12->11, 51->10, 35->10, 34->10, 96->10, 15->9, 4->9, 23->9, 108->9, 13->9, 46->9, 20->9, 18->8, 48->8, 92->8, 33->8, 93->8, 69->8, 8->7, 16->7, 80->7, 27->7, 21->7, 74->7, 29->7, 52->7, 26->6, 135->6, 2->6, 57->6, 5->6, 19->5, 40->5, 121->5, 85->4, 24->3, 111->2, 62->2, 95->2, 123->2, 91->2, 140->2, 54->2, 128->2, 124->2, 125->2, 55->1, 148->1, 141->1, 132->1, 110->1
rootkit	130->3, 19->2, 135->1, 105->1, 117->1, 1->1, 24->1
satan	115->13869, 71->1346, 130->159, 63->140, 145->118, 57->27, 123->23, 22->11, 124->11, 59->9, 48->7, 135->7, 19->6, 82->6, 80->6, 91->6, 69->6, 81->5, 40->4, 93->4, 15->3, 94->3, 79->3, 4->3, 51->3, 30->3, 43->3, 72->3, 16->3, 67->3, 50->3, 113->3, 12->3, 17->3, 96->3, 92->3, 33->3, 28->3, 102->3, 60->3, 84->3, 121->3, 78->3, 42->3, 56->3, 100->2, 75->2, 8->2, 23->2, 18->2, 35->2, 108->2, 98->2, 27->2, 44->2, 34->2, 13->2, 21->2, 74->2, 29->2, 46->2, 20->2, 52->2, 25->2, 122->1, 26->1, 112->1, 49->1, 2->1, 73->1, 37->1, 10->1, 5->1
smurf	6->2807607, 124->279
spy	19->1, 116->1
teardrop	137->970, 77->9
warezclient	24->705, 144->274, 129->19, 64->14, 146->2, 130->2, 75->1, 145->1, 112->1, 142->1
warezmaster	24->18, 90->1, 129->1

Al poder comparar las evidencias agrupadas con los datos originales, que en este caso están etiquetados, otro parámetro interesante es el número de etiquetas que tiene asignado cada uno de los clústeres, que es el último paso que se realiza en el código 5.6. El resultado se muestra en la tabla 5.2.

Finalmente, para concluir el experimento, de cara a detectar los datos más anómalos se pueden buscar las 100 mayores distancias y coger la más alejada. Filtrando los datos en base a la distancia con el centroide, en el caso de estudio salen aquellas evidencias que se alejan de los centroides más de lo esperado, así pues **anomalías**. Se puede ver el código y el listado resultante en el siguiente fragmento de código 5.7.

Código 5.7: Anomalías en KDDCup

```
//Anomaly Detection
val distances = normalizeddata.map{ case(label, datum) => distToCentroid(datum, model)}
val threshold = distances.top(100).last

val anomalies = normalizeddata.filter{ case(label, datum) => distToCentroid(datum,
    model) > threshold}
anomalies.keys.countByValue().foreach(println)

//res33: scala.collection.Map[String,Long] = Map(satan. -> 1, loadmodule. -> 2,
    multihop. -> 2, warezmaster. -> 1, portsweep. -> 6, perl. -> 3, ftp_write. -> 2,
    spy. -> 1, guess_passwd. -> 1, normal. -> 79, rootkit. -> 1)
```

5.3 Clasificación.

Las técnicas denominadas como regresión se centran en la predicción de un valor numérico en referencia a los valores anteriores, mientras que las técnicas de clasificación se ocupan en la predicción de una categoría. Las técnicas de clasificación y de regresión son con diferencia las más antiguas y más estudiadas en el área de la Predición Analítica. Apache Spark, mediante la librería de cálculo orientada a *Machine Learning* ofrece los siguientes algoritmos de clasificación:

- Clasificación binaria:
  - SVM lineales.
  - Regresión Logística.
  - Árboles de Decisión.
  - Random Forest.
  - Gradient-Boosted Trees.
  - Naïve Bayes.
- Clasificación Multiclase:
  - Regresión Logística.
  - Árboles de decisión.
  - Random Forest.
  - Naïve Bayes.
- Regresión:

- Mínimos cuadrados lineales.
- Lasso.
- Regresión Rigde.
- Árboles de decisión.
- Random Forest.
- Gradient-Boosted Trees.
- Regresión Isotonic.

5.3.1 **Métricas de sistemas de clasificación.**

Para poder realizar la evaluación de cada uno de los modelos anteriores aplicados a la clasificación y saber cómo de bien o de mal funcionan, como ejemplo, en un clasificador binario, donde la clase 1 es la relevante, se tendrán los siguientes datos:

- TP (*true positive*): Subconjunto de todos los datos predecidos como clase 1 que pertenecen a la clase 1.
- FP (*false positive*): Subconjunto de todos los datos predecidos como clase 1 que pertenecen a la clase 0.
- FN (*false negatives*): Subconjunto de todos los datos predecidos como clase 0 que pertenecen a la clase 1.
- TN (*true negatives*): Subconjunto de todos los datos predecidos como clase 0 que pertenecen a la clase 0.

A modo de recordatorio, los términos negativos o positivos hacen referencia a lo que estima el clasificador, mientras que el *true* o *false* hace referencia al valor real que tiene en la evidencia conocida, figura 5.4. Estos cuatro conceptos habitualmente se introducen en lo que se denomina la "matriz de confusion" [2x2](también llamada *contingency table* ó *confusion matrix*).

Cuadro 5.4: Tabla explicativa de TP, FP, FN y TN.

Población Total	Positivos Reales	Negativos Reales
Test Data - Resultados Positivos	TP	FP
Test Data - Resultados Negativos	FN	TN

Las métricas que se usan habitualmente en clasificación binaria son:

- Acierto (*Accuracy*) y error. El acierto es el número de muestras de entrenamiento que han sido bien clasificadas dividido por el total de las muestra de ejemplo, ecuación 5.7. El error es el número de muestras de entrenamiento que han sido mal clasificadas, dividido por el total de muestras de ejemplo, ecuación 5.8.

$$Accuracy\ (ACC) = \frac{TP + TN}{Total} \tag{5.7}$$

$$Error = \frac{FP + FN}{Total} \tag{5.8}$$

- Precisión y *recall*. La precisión se define como el número de muestras correctamente clasificadas (TP) dividido por la suma del número total de muestras correctamente clasificadas (TP) más el número de muestras mal clasificadas como positivas (FP). Se entiende como *Recall* al número de muestras correctamente clasificadas (TP), dividido de la suma de muestras correctamente clasificadas (TP) tanto positivas como negativas.

$$Precision = \frac{TP}{TP + FP} \tag{5.9}$$

$$Recall = \frac{TP}{TP + FN} \tag{5.10}$$

- Área por debajo de la curva precision-recall. Es típico generar para un clasificador la curva Precisión - Recall, haciendo que el área debajo de esta curva se denomine precisión media. Se calcula variando las opciones del clasificador.

- Curva ROC. Es un concepto parecido a la curva PR, representando la tasa de aciertos reales frente a la tasa de falsos aciertos reales. TPR, o sensibilidad, es el número de aciertos positivos reales (TP) dividido entre la suma de TP y falsos negativos (FP), ecuación 5.11. FPR, o *fall-out*, es el número de falsos positivos (FP) dividido entre la suma de FP más TN (número de ejemplos correctamente predecidos como otra clase), ecuación 5.12.

$$TPR = \frac{TP}{Positive} = \frac{TP}{TP + FN} \tag{5.11}$$

$$FPR = \frac{FP}{Negative} = \frac{FP}{FP + TN} \tag{5.12}$$

- AUC, Área por debajo de la curva ROC, representa un valor medio, donde 1.0 será el clasificador perfecto y 0.5 es el clasificador aleatorio.

Apache Spark MLlib [Fou15b] viene con varias clases de ayuda para la obtención de los parámetros anteriores, siendo diferentes para clasificadores binarios que para clasificadores multiclase.

Dado que como se ha comentado se tienen 23 categorías, se va a hacer uso de las técnicas de Clasificación Multiclase. La adaptación de los datos se hará de forma muy parecida al caso de clusterización, sólo que en vez de usar la codificación *one-hot* se indicará debidamente que determinadas características deberán ser entendidas como categorías y no de forma numérica. En total existen las siguientes categorías en el DataSet:

- Protocolos: 3 posibles valores.
- Servicios: 70 posibles valores.
- Estados TCP: 11 posibles valores.
- Clasificaciones: 23 posibles valores.

5.3.2 **Decision Trees y Random Forest**

La familia de algoritmos conocida como Árboles de Decisión, *Decision Trees* o su extensión los denominados *Random Forest* pueden tomar tanto características numéricas como categóricas, y su implementación en paralelo es relativamente sencilla, por lo que son de los primeros algoritmos que se pusieron a funcionar en entorno de computación paralela, como Apache Spark [Fou15b].

Los árboles de decisión son un algoritmo que particiona binariamente el espacio de dimensiones del *DataSet* en estudio. El árbol que se forma predice la misma categoría para toda la rama hijas incluidas. Cada partición se escoge seleccionando la mejor forma de trocear el espacio de todos los posibles cortes, de forma que se maximice la ganancia de la información en el nodo final, ecuación 5.16

La impureza de un nodo es la medida de cómo de homogéneas son las etiquetas del nodo, y la implementación actual de Spark MLlib de cara a mutl-clasificadores dispone de dos medidas de impureza:

- Gini, en la ecuación 5.13. Donde f_i es la frecuencia de la etiqueta i en un nodo y C es el número de etiquetas únicas.

$$\sum_{i=1}^{C} f_i(1 - f_i) \tag{5.13}$$

- Entropía, en la ecuación 5.14. Donde f_i es la frecuencia de la etiqueta i en un nodo y C es el número de etiquetas únicas.

$$\sum_{i=1}^{C} -f_i \log(f_i) \tag{5.14}$$

La denominada *Ganancia de Información*, IG, es la diferencia entre la impureza, I, del nodo padre y el promedio de las impurezas de sus dos nodos hijo. Asumiendo que un corte s particiona el *DataSet* D en dos *DataSets* D_l y D_r (*left y right*) de tamaños N_l y N_r, la ganancia de información vendría representada en la ecuación 5.15.

$$IG(D,s) = I(D) - \frac{N_l}{N} I(D_l) - \frac{N_r}{N} I(D_r) \tag{5.15}$$

$$\forall s \implies \arg max_s IG(D,s) \tag{5.16}$$

A la hora de realizar el corte del *DataSet*, se debe diferenciar las características que son continuas de las que son categóricas. A este respecto, la pequeños *DataSets* en implementaciones locales, los candidatos de corte para cada características continua son típicamente los valores únicos de la característica. Algunas implementaciones ordenan los valores de las características, y de esta forma utilizan esos vectores ordenados como candidatos de corte para cálculos más rápidos. La ordenación en entornos distribuidos es muy costosa computacionalmente, así que las implementaciones en estos entornos se basan en un conjunto aproximado de candidatos de corte haciendo un cálculo parcial de una porción de los datos. Los cortes ordenados crean los denominados *bins* y generalmente es un parámetro que se especifica en el lanzamiento del algoritmo. Denotar que obviamente el número de *bins* nunca puede ser superior al número de instancias N y que el propio algoritmo reduce el número de éstos si no se satisface la condición de corte.

Para las características categóricas con M posibles valores, se puede pensar en $2^{M-1} - 1$ candidatos de corte, que en clasificación y regresión binaria

se puede reducir a $M-1$ candidatos ordenando los valores de la característica categórica por la característica media. En los clasificadores multiclase, el total de $2^{M-1}-1$ cortes son posibles, asignando a cada uno su impureza y ordenándolos en base a ella.

La regla de parada acaba en un nodo cuando se cumple alguna de las siguientes premisas:

- La profundidad del mismo es igual a la máxima establecida (*maxDepth*).
- No hay corte posible que genere ganancia de información superior a la especificada (*minInfoGain*).
- No hay corte posible que produzca hijos hoja que tengan el número mínimo de instancias por nodo (*minInstancesPerNode*).

Los denominados ***Random Forest*** son abstracciones de los Árboles de Decisión y representan uno de los modelos de *machine-learning* más satisfactorios para clasificación y regresión. Se basan en combinar varios Árboles de Decisión de forma que se trate de reducir el *overfitting*. Al igual que los árboles de decisión, los *Random Forest* tratan con características categóricas, cumplen con clasificación multiclase, no requieren de escalado de categorías y son capaces de encontrar interacciones entre las características y no-linealidades.

Básicamente el algoritmo de *Random Forest*, al ser una abstracción de los árboles de decisión y contener varios de ellos en su interior, puede realizar un aprendizaje en paralelo, lo que facilita mucho el trabajo en entornos de computación paralela. El algoritmo además genera cierta aleatoriedad en el proceso de aprendizaje, haciendo que cada uno de los árboles que tiene en su interior sea diferente. Combinando las predicciones de cada uno de los árboles que tiene en su interior se reduce la variación en la predicción final, mejorando considerablemente el rendimiento en los datos de ensayo. La aleatoriedad en el proceso de aprendizaje viene dada por:

- *Bootstrapping*, subsampleo del *DataSet* original en cada iteración para obtener un juego de entrenamiento diferente.
- Se consideran subconjuntos aleatorios diferentes de características para cortar cada nivel del árbol.

A la hora de hacer una predicción ante una nueva instancia (vector de evidencias), el *Random Forest* debe agregar las predicciones que realizan

cada uno de sus árboles de decisión internos. La agregación cuando se trata de clasificación es por el "voto de la mayoría", aquella etiqueta que tenga más votos será el resultado final; mientras que en el caso de la regresión se realiza aplicando el estimador estadístico media.

5.3.3 **Naïve Bayes.**

El algoritmo de Naïve Bayes es un algoritmo simple multiclase que asume la independencia entre cada par de características. Naïve Bayes ofrece una fase de entrenamiento muy eficiente, que con una sola pasada sobre el *DataSet* de entrenamiento calcula la distribución de probabilidad condicional de cada etiqueta dada en las características, aplicando el Teorema de Bayes para calcular la distribución condicional de probabilidad de una etiqueta determinada dada una observación, por lo que se usa para la predicción.

MLlib soporta dos tipos: multinomial y Bernoulli, y generalmente son usados para la clasificación de documentos. En ese contexto, cada observación es un documento, y cada característica representa un término cuyo valor es la frecuencia de aparición del mismo, en Bayes multinomial; o un cero o un uno indicando si el término ha sido encontrado en el documento, en Bernoulli. Es por ello que los valores no pueden ser negativos al introducirlos en el modelo. El parámetro que se configura es el denominado *Additive smoothing*, representado por el parámetro λ y se debe tener en cuenta que ya que sólo se usa el *DataSet* una vez, éste no tiene necesidad de ser cacheado.

En el experimento que se va a realizar, se va a hacer uso de Bayes Multinomial, donde los vectores de características toman la forma de las frecuencias con las que se dan ciertos eventos, donde el multinomial generado es $(p_1, ..., p_n)$ siendo p_i la probabilidad de que el evento ocurra. Un vector de características $x = (x_1, ..., x_n)$ es por tanto un histograma, con x_i siendo el contador del número de veces que el evento i ha sido observado en un instante determinado. La semejanza (*likelihood*) de un vector x determinado es dado por la ecuación 5.17.

$$P(x/C_k) = \frac{(\sum_i x_i)!}{\prod_i x_i!} \prod_i p_{k_i}^{x_i} \tag{5.17}$$

El clasificador Naïve Bayes multinomial se vuelve un clasificador lineal cuando se expresa en espacio logarítmico, ecuación 5.18.

$$\log p(C_k/x) \quad \propto \quad log\left(p(C_k)\prod_{i=1}^{n} p_{k_i}\right) = \log p(C_k) + \sum_{i=1}^{n} x_i \ \log p_{k_i} = b + w_k^T x \tag{5.18}$$

Si una determinada clase y valor del vector de características no ocurren nunca a la vez en los datos de entrenamiento, la probabilidad estimada será de 0. El parámetro λ controla el *Additive Smoothing* que es una técnica por la que dada una observación $x = (x_1, ..., x_d)$ de una distribución multinomial con N ejemplos se puede crear una versión mejorada usando el estimador de la ecuación 5.19. Donde α es el parámetro con el que se suavizará.

$$(i = 1, ..., 3) \quad \Theta_i = \frac{x_i + \alpha}{N + \alpha_d} \quad \Longrightarrow \quad \Theta = (\Theta_1, ..., \Theta_d) \tag{5.19}$$

Desde un punto de vista Bayesiano, este parámetro corresponde con el valor estimado de la distribución posterior, usando una distribución Dirichlet que tiene α como apriori.

5.3.4 Experimentación y análisis de resultados Decision Trees, Random Forest y Naive Bayes.

El proceso a seguir es el siguiente:

1. El primer paso que se realiza es la importación de los datos, al igual que en el caso de clusterización por medio de la función indicada en el código 5.8. Se codifica la propia etiqueta de cada caso a un número entero, a diferencia del vector generado para clusterización. El dato a generar serán *LabeledPoints*, que recibe el número de la etiqueta y el vector correspondiente.

Código 5.8: Carga KDDCup para clasificación.

```
def buildDataSet(rawData: RDD[String]): (String => (String,LabeledPoint)) = {
      val splitData = rawData.map(_.split(','))
      val protocols = splitData.map(_(1)).distinct().collect().zipWithIndex.toMap
      val services = splitData.map(_(2)).distinct().collect().zipWithIndex.toMap
      val tcpStates = splitData.map(_(3)).distinct().collect().zipWithIndex.toMap
      val labels = splitData.map(l => l(l.length
           -1)).distinct().collect().zipWithIndex.toMap

      (line: String) => {
            val buffer = line.split(',').toBuffer
            val protocol = buffer.remove(1)
            val service = buffer.remove(1)
            val tcpState = buffer.remove(1)
```

```
                val label = buffer.remove(buffer.length -1)
                val vector = buffer.map(_.toDouble)

                val newProtocolFeatures = new Array[Double](protocols.size)
                newProtocolFeatures(protocols(protocol)) = 1.0
                val newServicesFeatures = new Array[Double](services.size)
                newServicesFeatures(services(service)) = 1.0
                val newTcpStateFeatures = new Array[Double](tcpStates.size)
                newTcpStateFeatures(tcpStates(tcpState)) = 1.0

                vector.insertAll(1, newTcpStateFeatures)
                vector.insertAll(1, newServicesFeatures)
                vector.insertAll(1, newProtocolFeatures)
                (label, LabeledPoint(labels(label), Vectors.dense(vector.toArray)))
        }
}
```

2. Se hacen hacer 3 subconjuntos aleatorios de los datos para generar los grupos: *training*, *cross-validation* y *testing*, código 5.9. Los tres grupos, para evitar que se recalculen constantemente serán cacheados en el driver y en los ejecutores que los usen la primera vez que se computen.

Código 5.9: Lectura de datos y creación de Train CV y Test.

```
...
val data = rawData.map(buildDataSet(rawData))

val Array(trainData, cvData, testData) = data.randomSplit(Array(0.8,0.1,0.1))
trainData.cache()
cvData.cache()
testData.cache()
...
```

3. Se definen las funciones de evaluación para cada algoritmo, código 5.10. No es el caso en los que se está usando pero alguna implementación en MLlib require de fijar thresholds por encima de un valor de cara a calcular las métricas de cada modelo.

Código 5.10: Conteo de total de casos por categoría

```
def getMetrics(model: DecisionTreeModel, data: RDD[LabeledPoint]): MulticlassMetrics
    = {
  val predictionsAndLabels = data.map { example =>
        (model.predict(example.features), example.label)
  }
  new MulticlassMetrics(predictionsAndLabels)
}

def getMetricsForest(model: RandomForestModel, data: RDD[LabeledPoint]):
    MulticlassMetrics = {
  val predictionsAndLabels = data.map { example =>
```

```
        (model.predict(example.features), example.label)
  }
  new MulticlassMetrics(predictionsAndLabels)
}

def getMetricsNB(model: NaiveBayesModel, data: RDD[LabeledPoint]): MulticlassMetrics
    = {
  val predictionsAndLabels = data.map { example =>
        (model.predict(example.features), example.label)
  }
  new MulticlassMetrics(predictionsAndLabels)
}
```

4. Se lanza la búsqueda de los parámetros más adecuados para los tres modelos. En cada uno de ellos se varían los parámetros sobre los que se calcula el modelo para evaluar su precisión y su exhaustividad (*recall*). Se sacan los valores anteriores tanto para el grupo de datos de entrenamiento, como para el grupo de datos de cros-validación. Cómo sólo se aprende el modelo con los datos de entrenamiento y los de cros-validación no son conocidos, sirve esta última métrica para poder evaluar la bondad del modelo con los parámetros seleccionados. Todo esto se puede ver en el código 5.11

Código 5.11: Búsqueda Parámetros

```
...
val evaluationsTree =
      for(impurity <- Array("gini", "entropy");
            depth  <- Array(10,20,30);
            bins   <- Array(70,150,300))
                  yield{
                        val model = DecisionTree.trainClassifier(trainData.values,
                              23, Map(1 -> 3, 2 -> 70, 3 -> 11), impurity, depth,
                              bins)
                        val trainAccuracy = getMetrics(model, trainData.values)
                        val cvAccuracy = getMetrics(model, cvData.values)
                        "" + impurity +","+ depth+","+bins +"," +
                              trainAccuracy.precision.toString() + "," +
                              trainAccuracy.recall.toString() + "," +
                              cvAccuracy.precision.toString() +","+
                              cvAccuracy.recall.toString()
                  }
sc.parallelize(evaluationsTree.toSeq).saveAsTextFile(args(1).toString())

val evaluationsForest =
      for(impurity <- Array("gini", "entropy");
            depth  <- Array(10,20,30);
            bins   <- Array(70,150,300))
                  yield{
                        val model = RandomForest.trainClassifier(trainData.values,
                              23, Map[Int,Int](), 10, "auto", impurity, depth, bins)
                        val trainAccuracy = getMetricsForest(model,
                              trainData.values)
```

```
                        val cvAccuracy = getMetricsForest(model, cvData.values)
                        "" + impurity +","+ depth+","+bins +"," +
                             trainAccuracy.precision.toString() + "," +
                             trainAccuracy.recall.toString() + "," +
                             cvAccuracy.precision.toString() +","+
                             cvAccuracy.recall.toString()
                  }
sc.parallelize(evaluationsForest.toSeq).saveAsTextFile(args(2).toString())

val evaluationsNB =
      for(modelType <- Array("multinomial"/*,"bernoulli"*/); //Bernoulli requires 1
           or 0 in features
             lambda <- Array(0.1, 0.5, 1.0, 2.0, 10.0, 20.0, 50.0, 100.0, 200.0,
                  500.0, 1000.0))
                yield {
                        val model = NaiveBayes.train(trainData.values, lambda,
                             modelType)
                        val trainAccuracy = getMetricsNB(model, trainData.values)
                        val cvAccuracy = getMetricsNB(model, cvData.values)
                        "" + modelType +","+ lambda+"," +
                             trainAccuracy.precision.toString() + "," +
                             trainAccuracy.recall.toString() + "," +
                             cvAccuracy.precision.toString() +","+
                             cvAccuracy.recall.toString()
                  }
sc.parallelize(evaluationsNB.toSeq).saveAsTextFile(args(3).toString())
...
```

Los resultados son exportados a HDFS de forma que se puedan analizar externamente. Y las figuras 5.6, 5.7 y 5.8, muestran los resultados tanto para el grupo de *training*, como para el grupo de *cross-validation*.

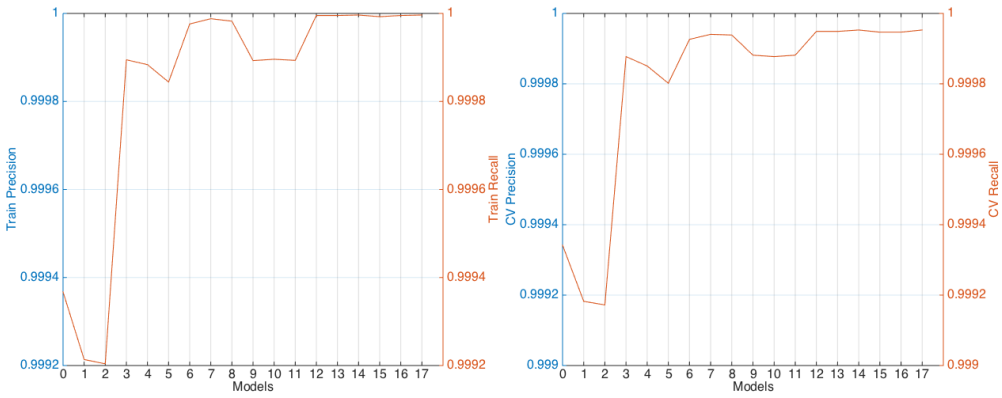


Figura 5.6: Aprendizaje Decision Tree.

Los ejes de las coordenadas son los diferentes modelos que han sido probados, y por tanto, se seleccionará en las gráficas de PR para los datos de *cross-validate* aquellos modelos que hagan máximo los

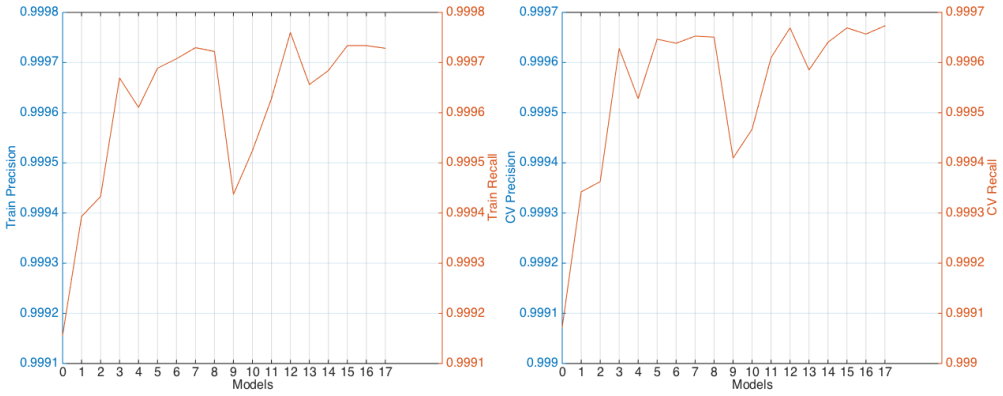


Figura 5.7: Aprendizaje Random Forest.

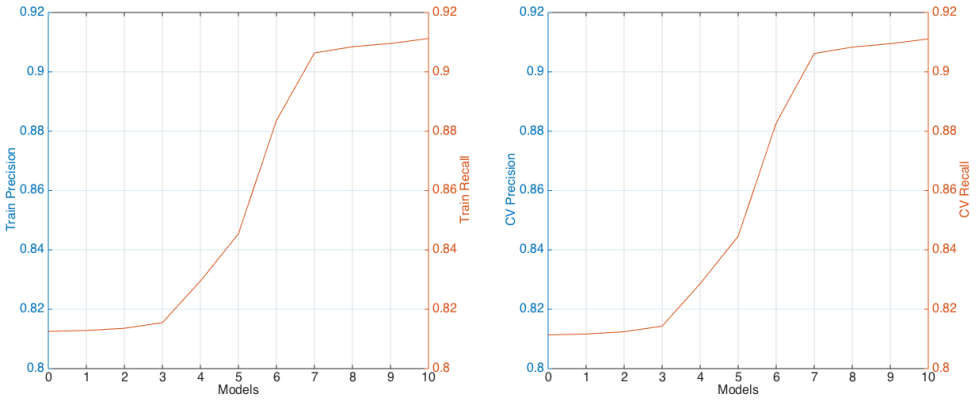


Figura 5.8: Aprendizaje Naive Bayes.

parámetros. En todas las gráficas el parámetro de precisión y de exhaustividad ha sido el mismo y van solapados. Los mejores modelos son:

- Decision Tree: ID 17, que corresponde con el modo de impureza "entropy", máxima profundidad de 30 y un *bins* de 300.
- Random Forest: ID 5, que corresponde con el modo de impureza "entropy", máxima profundidad de 30 y un *bins* de 300.
- Naïve Bayes, el mejor es el último modelo, fijando $\lambda = 1000{,}0$.

5. Se cogen los mejores parámetros de cada uno de los modelos, se juntan en la base de entrenamiento los grupos de *training* y *cross-validation* y se prueban los resultados finales contra el grupo de test, código 5.12. Los resultados finales se pueden ver en la tabla 5.5.

Código 5.12: Conteo de total de casos por categoría

```
...
val treeModel = DecisionTree.trainClassifier(trainData.union(cvData).values, 23,
     Map(1 -> 3, 2 -> 70, 3 -> 11), "entropy", 30, 300)
val testTreeAccuracy = getMetrics(treeModel, testData.values)
println("" +testTreeAccuracy.precision.toString() + "," +
     testTreeAccuracy.recall.toString())

val forestModel = RandomForest.trainClassifier(trainData.union(cvData).values,23,
     Map(1 -> 3, 2 -> 70, 3 -> 11), 10, "auto", "gini", 20, 150)
val testForestAccuracy = getMetricsForest(forestModel, testData.values)
println("" +testForestAccuracy.precision.toString() + "," +
     testForestAccuracy.recall.toString())
testForestAccuracy.confusionMatrix

val naiveModel = NaiveBayes.train(trainData.union(cvData).values, 1000.0,
     "multinomial")
val testNaiveAccuracy = getMetricsNB(naiveModel, testData.values)
println("" +testNaiveAccuracy.precision.toString() + "," +
     testNaiveAccuracy.recall.toString())
...
```

Cuadro 5.5: Resultados finales Clasificadores.

Modelo	Precission	Recall
Decision Tree	0.9999388811584873	0.9999388811584873
Random Forest	0.9997188533290414	0.9997188533290414
Naïve Bayes	0.9121864858092237	0.9121864858092237

5.4 Amazon EC2. Escalabilidad de algoritmia paralela.

El siguiente apartado, usando las pruebas anteriores, tratará de evidenciar la escalabilidad de los algoritmos a nivel temporal con un diferente número de máquinas asignadas al clúster en cuestión, usando para ello la computación en la nube que ofrecen los servicios de Amazon EC2 [Ama15a].

5.4.1 Lanzamiento en Amazon EC2

La distribución de Apache Spark, bien compilada, bien desde fuentes, dispone de un script específico para controlar máquinas virtuales en Amazon EC2: crear y lanzar el clúster, gestionar el mismo y finalizarlo. Automáticamente se crea tanto la distribución de Spark que estará desplegada en el clúster, como el slstema de ficheros distribuidos HDFS del mismo. Para todo es necesario disponer de una cuenta de Amacon Web Services [Ama15b], como se muestra en la figura 5.9.

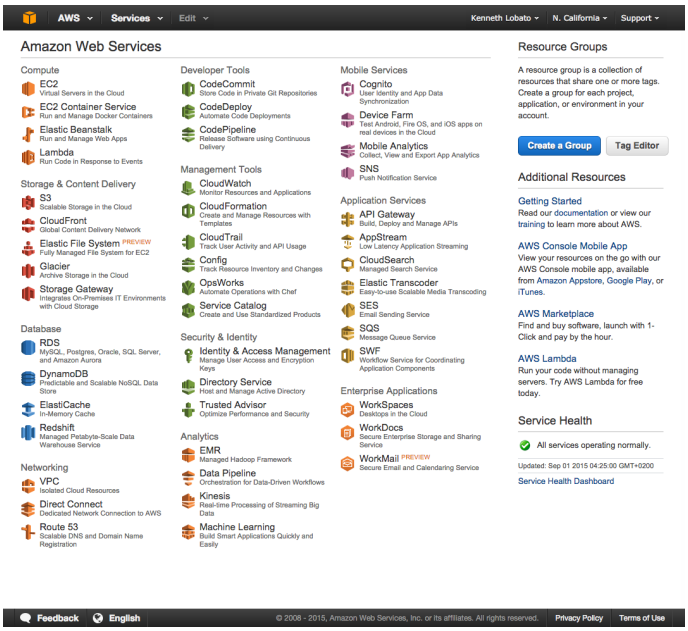


Figura 5.9: Amazon AWS Management Console.

El script que se usa para ello se denomina *spark-ec2*, y está diseñado para gestionar múltiples clústeres por nombre. El nombre que se le asigna al clúster es el que se deberá usar para logearse en el mismo. Todos los clústeres que se creen se identifican por medio del grupo de seguridad que se crea en la cuenta de EC2, asignando los nombres a las máquinas en función de nombre del clúster. Los pasos previos a realizar son los siguientes:

- Crear un nuevo key-pair. De esta forma haremos que el ssh al clúster pueda funcionar con el sistema PKI de claves públicas-privadas.
- Obtener las claves de acceso: AWS ACCESS KEY y AWS SECRET ACCESS KEY

- Conocer los identificadores apropiados para la zona donde se desean las máquinas virtuales, así como las redes que se deben asociar a las mismas.

Una vez lanzado el clúster, se puede acceder por medio de un navegador web como se observa en la figura 5.10. Pudiendo ya lanzar los trabajos que se deseen en el clúster. Los comandos para lanzar la gestión de las máquinas del clúster están indicados en el listado de código 5.13.

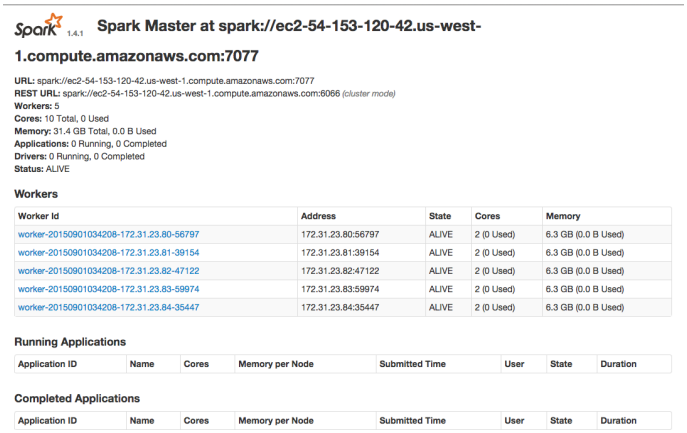


Figura 5.10: Amazon AWS Management Console.

Código 5.13: Gestión de SPARK en EC2.

```
MacBook-Pro-de-Kenneth:thesisKlobato klobato$ ./spark-ec2 --key-pair=klobatoThesisKey
     --identity-file=./klobatoThesisKey.pem --region=us-west-1 --zone=us-west-1b
     --instance-type=c3.2xlarge --ebs-vol-size=100 --spark-version=1.4.1 -s 4
     --vpc-id=vpc-ce38e9ab --subnet-id=subnet-9dab1ff8 launch klobatoThesisCluster

MacBook-Pro-de-Kenneth:thesisKlobato klobato$ ./spark-ec2 --key-pair=klobatoThesisKey
     --identity-file=./klobatoThesisKey.pem --region=us-west-1 --zone=us-west-1b
     --instance-type=c3.2xlarge --ebs-vol-size=100 --spark-version=1.4.1 -s 4
     --vpc-id=vpc-ce38e9ab --subnet-id=subnet-9dab1ff8 launch klobatoThesisCluster --resume

MacBook-Pro-de-Kenneth:thesisKlobato klobato$ ./spark-ec2 --key-pair=klobatoThesisKey
     --identity-file=./klobatoThesisKey.pem --region=us-west-1 --zone=us-west-1b
     --instance-type=c3.2xlarge --ebs-vol-size=100 --spark-version=1.4.1 -s 4
     --vpc-id=vpc-ce38e9ab --subnet-id=subnet-9dab1ff8 login klobatoThesisCluster

MacBook-Pro-de-Kenneth:thesisKlobato klobato$ ./spark-ec2 --key-pair=klobatoThesisKey
     --identity-file=./klobatoThesisKey.pem --region=us-west-1 --zone=us-west-1b
     --instance-type=c3.2xlarge --ebs-vol-size=100 --spark-version=1.4.1 -s 4
     --vpc-id=vpc-ce38e9ab --subnet-id=subnet-9dab1ff8 stop klobatoThesisCluster
```

```
MacBook-Pro-de-Kenneth:thesisKlobato klobato$ ./spark-ec2 --key-pair=klobatoThesisKey
     --identity-file=./klobatoThesisKey.pem --region=us-west-1 --zone=us-west-1b
     --instance-type=c3.2xlarge --ebs-vol-size=100 --spark-version=1.4.1 -s 4
     --vpc-id=vpc-ce38e9ab --subnet-id=subnet-9dab1ff8 destroy klobatoThesisCluster
```

El clúster sobre el que se han hecho las pruebas esta compuesto inicialmente por máquinas tipo "c3.2xlarge", 1 de ellas hará de controlador principal y el resto serán los ejecutores que trabajarán con el *DataSet*. Las máquinas usadas en la nube tienen las siguientes características:

- 8 x Procesador Intel Xeon E5-2680 v2 (Ivy Bridge) de alta frecuencia.
- 15GB de memoria RAM.
- 2x80GB de disco duro SSD.

Después del lanzamiento de la orden de creación, el script se ocupa de descargar la versión deseada de Apache Spark y de configurar un sistema de archivos distribuidos HDFS. Es en ese sistema donde se copiará para los experimentos de este capítulo el fichero de *kddcup.data*.

5.4.2 Resultados de escalabilidad

Para la evaluación de los tiempos se hará uso de la consola *spark-shell*, y se lanzará el código mostrado en 5.14 para evaluar los tiempos de cálculo tanto de los modelos de clasificación como los modelos de clusterización. Los resultados de la prueba se pueden ver en la tabla 5.6 y en la figuras 5.11 y 5.12

Código 5.14: Código calculo temporal modelos en Amazon EC2.

```
...
       //Classifier:Decision Tree
       var t0 = System.nanoTime()
       val treeModel = DecisionTree.trainClassifier(trainData.union(cvData).values, 23,
            Map(1 -> 3, 2 -> 70, 3 -> 11), "entropy", 30, 300)
       var t1 = System.nanoTime()
       val testTreeAccuracy = getMetrics(treeModel, testData.values)
       var t2 = System.nanoTime()
       println("" +testTreeAccuracy.precision.toString() + "," +
            testTreeAccuracy.recall.toString()+"," + ((t1-t0)*1.0e-9).toString()+"," +
            ((t2-t1)*1.0e-9).toString())

       //Classifier: Random Forest
       t0 = System.nanoTime()
       val forestModel = RandomForest.trainClassifier(trainData.union(cvData).values,23,
            Map(1 -> 3, 2 -> 70, 3 -> 11), 10, "auto", "entropy", 30, 300)
       t1 = System.nanoTime()
       val testForestAccuracy = getMetricsForest(forestModel, testData.values)
```

```
        t2 = System.nanoTime()
        println("" +testForestAccuracy.precision.toString() + "," +
             testForestAccuracy.recall.toString()+"," + ((t1-t0)*1.0e-9).toString()+"," +
             ((t2-t1)*1.0e-9).toString())

        //Classifier: Naive Bayes
        t0 = System.nanoTime()
        val naiveModel = NaiveBayes.train(trainData.union(cvData).values, 1000.0,
             "multinomial")
        t1 = System.nanoTime()
        val testNaiveAccuracy = getMetricsNB(naiveModel, testData.values)
        t2 = System.nanoTime()
        println("" +testNaiveAccuracy.precision.toString() + "," +
             testNaiveAccuracy.recall.toString() +"," + ((t1-t0)*1.0e-9).toString()+"," +
             ((t2-t1)*1.0e-9).toString())
...
```

Cuadro 5.6: Resultados tiempos de escalabilidad Modelos en Amazon EC2.

Modelo	Tiempo Aprendizaje (s)			Tiempo Cálculo DataSet (s)		
Número de máquinas	5	10	20	5	10	20
Clasificador Decision Tree	57,94036	56,00984	61,91904	0,42687	0,43617	0,41207
Clasificador Random Forest	76,42565	73,84099	63.59639	0,41185	0,409614	0,40877
Clasificador Naïve Bayes	1,092404	1,10596	1.08457	0,36584	0,42585	0,40223
Clusterizador K-means	1190,42169	1038,95869	1091,10033	0,47388	0,43305	0,38900

Como conclusiones que se pueden extraer de la tabla 5.6 y las correspondientes figuras 5.11 y 5.12 se tiene lo siguiente:

- Los tiempos de aprendizaje de los clasificadores basados en Naïve Bayes son prácticamente iguales, con lo que el para el aprendizaje el tener varias máquinas no aporta. En cambio, de cara a los tiempos de evaluación contra el DataSet de *testing* sí que se nota una reducción.

- Para los algoritmos basados en árboles de decisión, se observa que el cálculo de los *Random Forest* en aprendizaje es superior al de los árboles normales, esto es debido a que se calculan 10 de los anteriores en cada modelo. El beneficio de disponer de más máquinas en aprendizaje de los modelos *Random Forest* queda patente, no siendo tan claro en el caso del modelo basado en *Decision Trees*. A nivel de evaluación del modelo con el *DataSet* de *testing* los resultados son muy semejantes, sin cambios significativos en el número de máquinas en el clúster.

- Finalmente, para los modelos K-means de clusterización o aprendizaje no supervisado, adolecen el mayor tiempo de todos en el aprendizaje, siendo plausible la mejora cuando se introducen más máquinas, pero llegando a un límite en el particionado de la información: puede

llegar un momento que se tarde más en distribuir en pequeñas porciones el *DataSet* para realizar el cálculo, que el tiempo que conlleve el cálculo en local del mismo. En la evaluación se observa que el clúster mejora los tiempos de respuesta conforme se aumenta el número de máquinas del mismo.

Denotar, que si bien las pruebas realizadas arrojan luz sobre la algoritmia y su comportamiento en un clúster de múltiples máquinas, el *DataSet* empleado no puede considerarse como un problema de *Big Data*, siendo un lote pequeño de información en cuanto al *volumen* real de datos para los que está concebida la tecnología empleada. En subsiguientes experimentos se rebasarán estos límites de cara a ofrecer una imagen más real de lo que *Big Data* puede llegar a ofrecer.

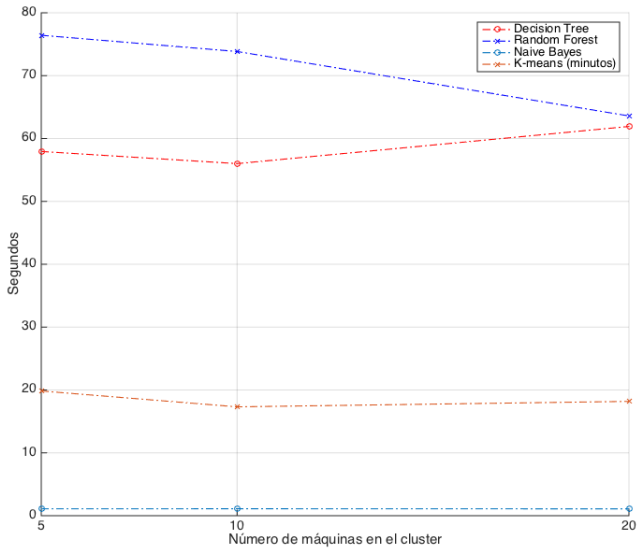


Figura 5.11: Resultados Tiempos Entrenamiento.

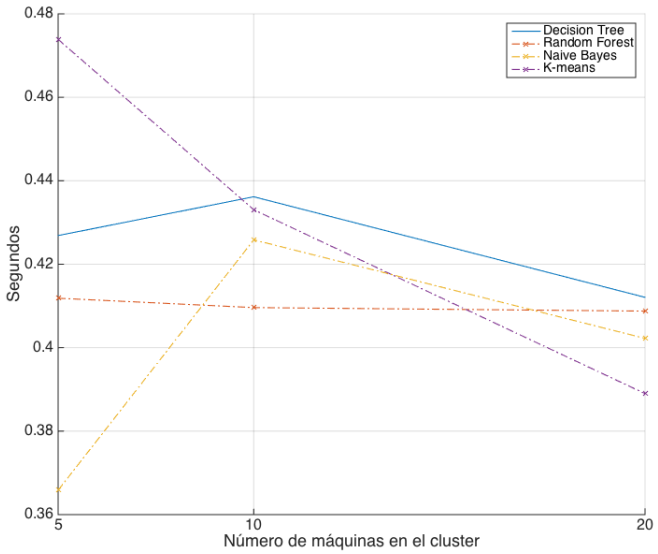


Figura 5.12: Resultados Tiempos Evaluación DataSet.

«A partir de cierto punto, en adelante no hay regreso. Es el punto que hay que alcanzar.»

Franz Kafka (1883-1924)

Capítulo

6

Recogida de evidencias en tiempo real.

La preparación del laboratorio de trabajo, así como los despliegues de las diferentes herramientas y desarrollos necesarios para hacer la recogida de evidencias centran el contenido del presente capítulo. Se comenzará por la descripción del laboratorio preparado, pasando a los mecanismos empleados tanto para realizar los ataques como para la recogida de evidencias de los diferentes equipos, haciendo los correspondientes experimentos para poder recoger y procesar la información generada.

Como se ha introducido en el capítulo 3, se parte de los SIEMs comerciales *open-source*, en este caso OSSIM de AlienVault. Este SIEM recibe su información principalmente de dos fuentes, una de *host* y otra de *network*: OSSEC [Mic15] y Snort [aiA15] respectivamente. En el presente capítulo, tomando como objetivo la integración número #2, presentada en la sección 4.11 del capítulo 4 de Arquitectura de ESIDE-BIDS, se realizará la experimentación correspondiente.

La adquisición seguirá las fases de la lógica representada en la figura 6.1. En ella se puede observar como las evidencias generadas por las fuentes de información, son traspasadas a un formato común estructurado (JSON). Este *buffer* será leído por los agentes de ESIDE-BIDS para enviar los mensajes con la información de los eventos a la capa de recepción montada sobre

Apache Kafka. Desde el clúster, haciendo uso de la tecnología Spark Streaming, se leerán los datos enviados a la capa de mensajería, persistiendo los mismos en formato CSV al sistema de archivos distribuido, HDFS.

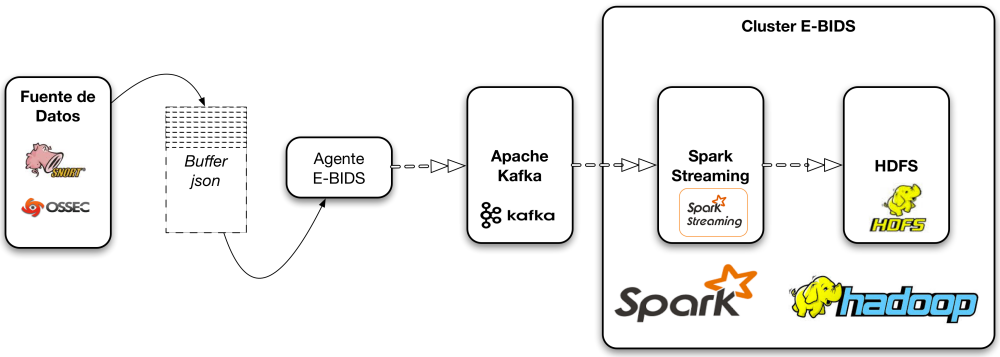


Figura 6.1: ESIDE-BIDS. Lógica de Adquisición.

A nivel experimental, como es el caso, toda la lógica se ha montado para que trabaje en formatos legibles y persistiendo la información en cada etapa, generalmente a ficheros CSV. Esto es para facilitar la experimentación, trazabilidad y la reproducibilidad de cada experimento, permitiendo optimización y mejoras, ya que la información recogida servirá de base para el capítulo 7 donde se analizará mediante algoritmia de *Machine Learning*. En sistemas productivos, lo habitual sería subir la información en formatos binarios comprimidos como AVRO [Fou15c] o Protocol Buffer [Dev15].

6.1 El laboratorio de trabajo.

Para poder acometer tanto el despliegue de los diversos sistemas operativos, herramientas y software necesario en la presente tesis se parte de los siguientes equipos:

- Equipo de sobremesa:
  - Placa base Asus Rampage IV Extreme Edition (integra tarjetas de red Gigabit).
  - Intel i7-3820 a 3.60GHz, 4 cores con 10 MB de caché L3.
  - 48 GB de RAM, DDR3 a 1866 MHz.

  - Almacenamiento: 2 discos SSD (Crucial SSD 512 GB y OCX Vertex 3 de 128 GB), 2 discos adicionales para persistencia, SEAGATE y SAMSUNG de 1 TB cada uno, a 7200 rpm.

- Equipo portátil: *Macbook Pro 15" 2014*
  - Intel Core i7 de 4 cores a 2,5 GHz con 6MB de caché L3.
  - 16 GB. RAM DDR3L integrada en placa base.
  - 512 GB de almacenamiento flash PCIe.
  - Red Wireless 802.11ac y red cableada gigabit.

- Equipo en la nube: *Amazon EC2 Instances*.
  - c3.2xlarge [1]

- Equipos de red:
  - Cableada: *Switch Netgear Gigabit GS108PE con alimentación PoE*.
  - Inalámbrica: *Apple AirPort Extreme 802.11ac*.

Para poder realizar la instalación y la puesta en marcha de todos los sistemas operativos necesarios para el laboratorio, se han usado tecnologías de VMware [Inc15]. En el equipo portátil se usa VMware Fusion, mientras en el equipo de sobremesa se ha instalado VMware ESXi 5.5, modificando los controladores de acceso a disco (DMA) y de red que vienen por defecto para ajustarse al modelo de la placa base. Para paliar la velocidad de procesamiento/respuesta de los grandes clústeres en comparación con el entorno de laboratorio se usan discos de estado sólido en aquellas imágenes que requieren computación intensiva, obteniendo resultados satisfactorios. El diagrama completo del queda reflejado en la figura 6.2.

A nivel general, las imágenes de Windows son de 64 bits con la versión indicada en su nombre. Todas las máquinas que tienen como nombre *Ubuntu*, se basan en *Ubuntu Server 14.04.3 LTS*. Así mismo, aquellas imágenes que tienen como nombre *Debian*, parten de *Debian 8.2 Jessie*. El resto de imágenes son las que los propios desarrolladores de esas tecnologías suministran, siempre en versiones de 64 bits.

Las máquinas Ubuntu1 a Ubuntu3, configuran el clúster de Hadoop (versión 2.6), desplegando varias versiones de Apache Spark (versión 1.3.1

[1] Definido en la sección 5.4

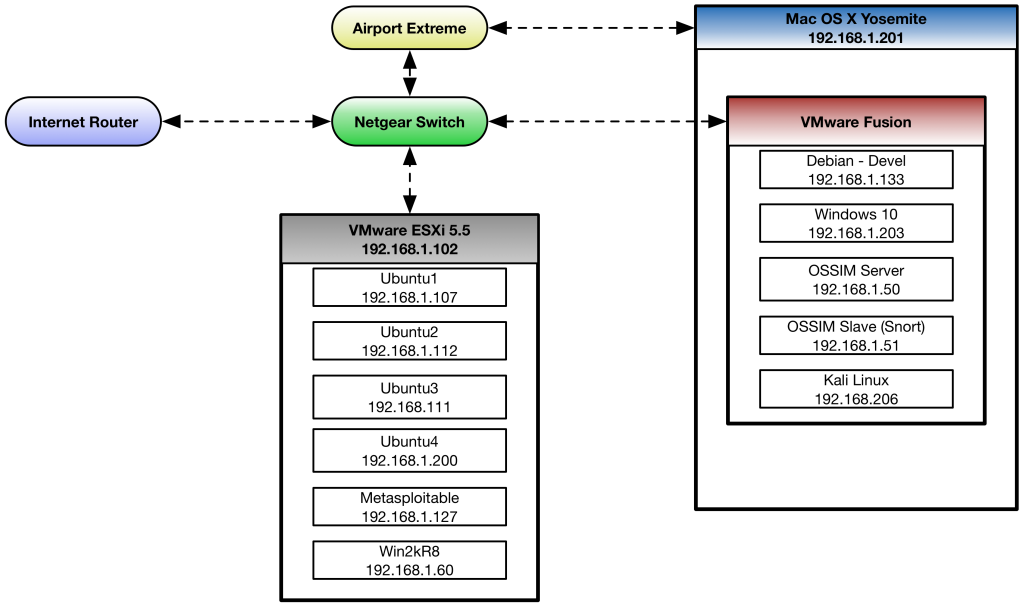


Figura 6.2: Diagrama del laboratorio usado en ESIDE-BIDS.

a 1.5.0) entre otros proyectos del ecosistema de Hadoop. La máquina Ubuntu4 se destina por completo al despliegue de Apache Kafka. Las imágenes Metasploitable y Windows Server 2008 R2 se usan como equipos a atacar. Las imágenes de OSSIM y agente OSSIM son el despliegue del SIEM de AlienVault, si bien las tecnologías que se usan para la recogida de evidencias que son Snort y OSSEC, se han trasladado a la imagen *Debian Devel*, para poder hacer modificaciones y desarrollos adicionales sobre ambos softwares. Finalmente, tanto el Mac OS X como la imagen de Windows 10, se han usado como sistema de escritorio y de trabajo.

6.2 Recogida de evidencias desde OSSEC.

6.2.1 Configuración de OSSEC.

Los agentes se han desplegado en todas las máquinas del laboratorio, tanto Windows como Linux, y envían la información al nodo central, que en este caso es la máquina denominada *Debian - Devel*. En la figura 3.7 del capítulo 3, sección 3.3, se puede ver el sistema desplegado y la información transferida mediante el proyecto *ossec-webui*.

A partir de la versión 2.9 de OSSEC, se puede hacer que el servicio central almacene la información en formato *json*, y este será el *buffer* de entrada que se utilizará para el envío de los eventos. La configuración del sistema se puede ver en el código 6.1.

Código 6.1: Configuración de Ossec para escritura de JSON.

```
...
  <global>
  <email_notification>yes</email_notification>
  <email_to>kenneth@klobato.net</email_to>
  <smtp_server>klobato-net.mail.protection.outlook.com.</smtp_server>
  <email_from>ossecm@debian-devel</email_from>
  <jsonout_output>yes</jsonout_output>
  </global>
...
```

El sistema una vez desplegado queda permanentemente conectado para recabar información que pueda usarse como tráfico limpio en el capítulo 7.

6.2.2 Análisis inicial de los datos de OSSEC.

OSSEC almacena la información como se puede ver en la figura 6.3. Sigue la misma filosofía que el resto de sistemas de logeo de los sistemas operativos UNIX, haciendo uso de la herramienta *logrotate*, para cambiar de log cuando termina el día, comprimiendo el mismo.

Para poder evaluar qué se tiene capturado como evidencias normales, se han juntado en un único fichero todas las evidencias *json*, dando como resultado un fichero de 649 MB. Se va a hacer uso de la funcionalidad de Spark SQL para procesar la información del fichero, para ello:

- Se carga el fichero en HDFS.
- Se procesa el JSON y se escribe en formato CSV.
- Se procesa el fichero CSV como si fuera un objeto de Spark (sección 4.10) denominado *DataFrame*.

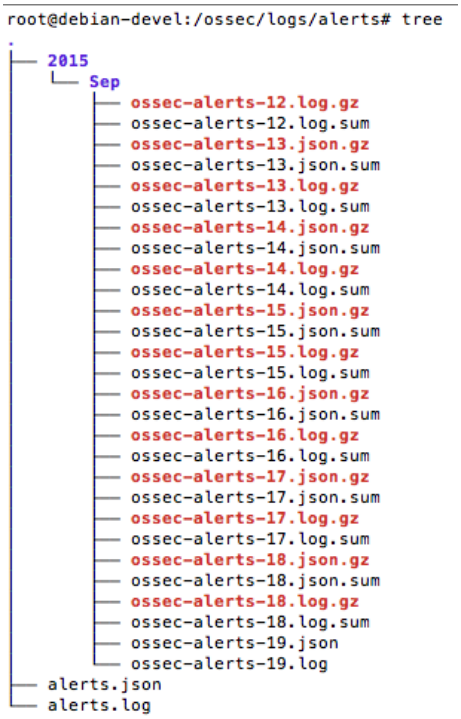


Figura 6.3: Organización interna de Alertas de OSSEC.

- Este objeto *DataFrame* permite la exportación a una tabla de Hive (sección 4.5).

Una vez exportada la información a Hive, mediante el servicio de *Thrift-Server* se puede dar acceso a estas tablas a través de un controlador ODBC o JDBC. Esto se usa para que se puedan usar en el análisis de datos motores de *Business Intelligence*, que en este caso concreto servirá para aplicar la herramienta ***Tableau*** [Sof15].

En un primer análisis de la información, que se puede ver en la figura 6.4, se aprecia que hay un número enorme de alarmas de tipo 20101, nivel 6, que acaban generando alarmas críticas en niveles del 8 al 10. Esto se explicará en la siguiente sección relativa a Snort, pero de momento se eliminan del análisis y se configura OSSEC para evitar esta información ya que llegaría por duplicado al sistema central de ESIDE-BIDS.

Una vez filtradas las alertas mencionadas relativas al NIDS, la distribución para la semana en estudio de tráfico limpio de Ossec, organizado por el nivel de alerta (de menor a mayor), por tipo, contando el número de máquinas en las que aparece y el total de cada alerta con su correspondiente agregado total, queda como se muestra en la figura 6.5.

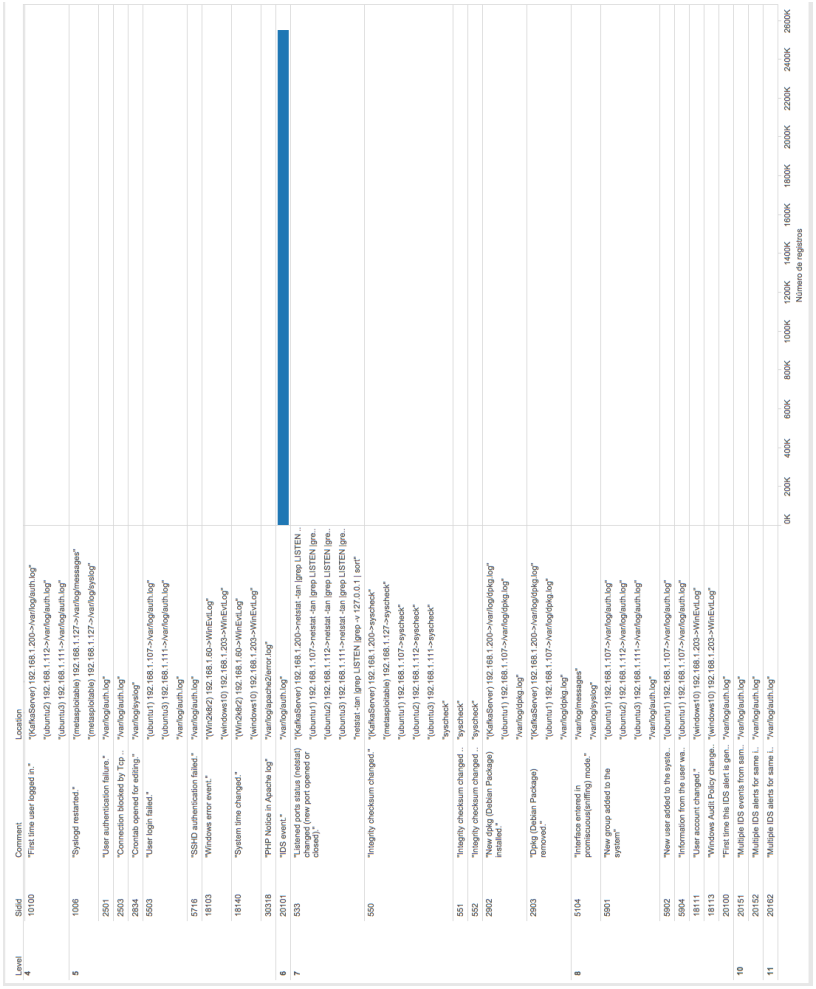


Figura 6.4: Análisis Ossec, con duplicados de alarmas de NIDS.

El volumen total de alertas generadas, de forma agregada es de 33.747, con los filtros indicados anteriormente, que estimando se produzcan todas en 8 horas de trabajo durante 5 días laborables:

$$Ev_T x \frac{1\ semana}{5\ dias}\ x\ \frac{1\ dia}{8\ horas}\ x\ \frac{1\ hora}{3600\ s} = 0,2343 \frac{Ev}{s} \tag{6.1}$$

La tasa mínima de transferencia que se debe garantizar para recoger las alarmas de OSSEC en tiempo real, como se puede apreciar en la ecuación 6.1 es relativamente baja, siendo inferior a un evento por segundo.

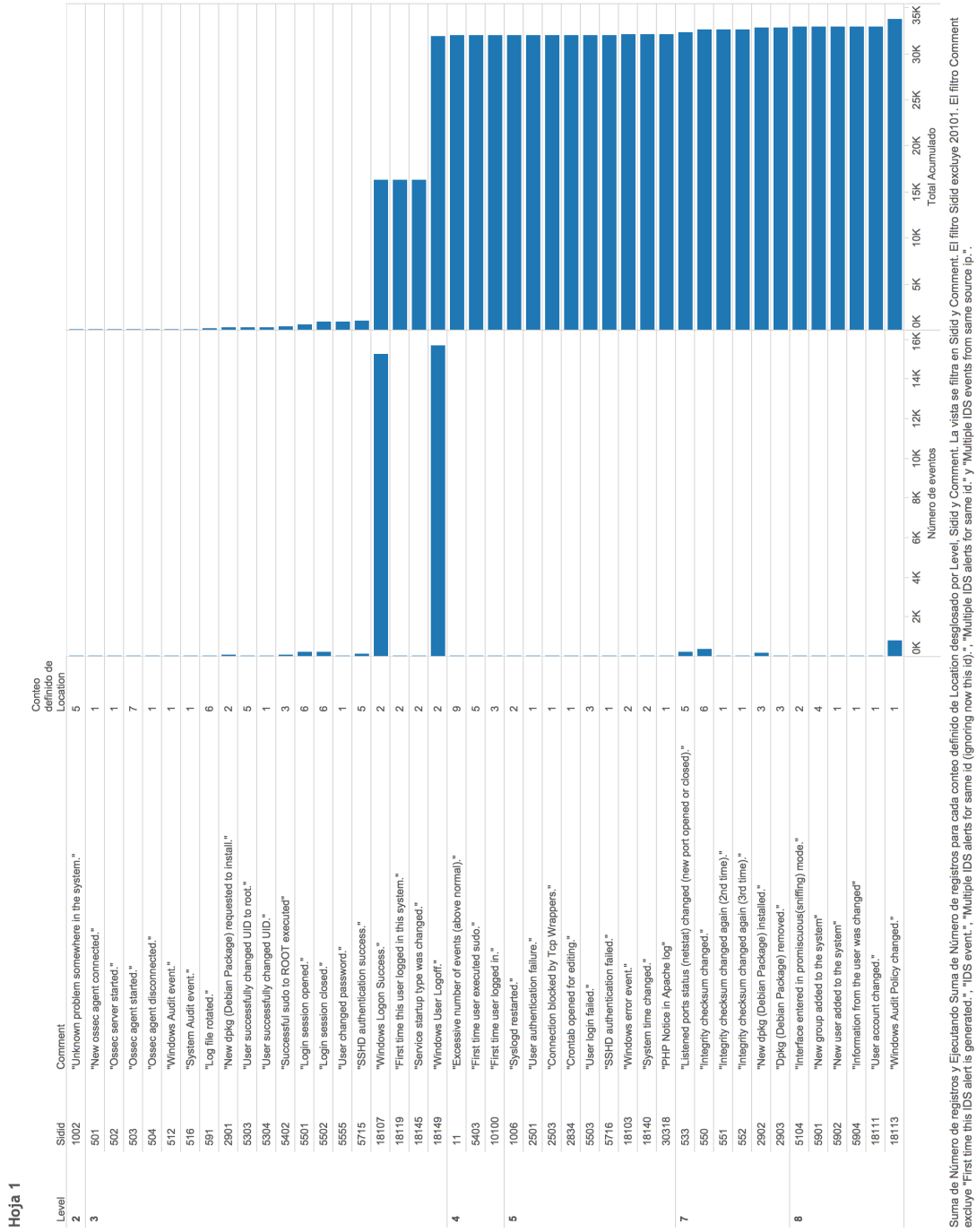


Figura 6.5: Análisis inicial de alarmas de Ossec.

6.2.3 Agente ESIDE-BIDS para OSSEC.

En concreto para esta tecnología se ha desarrollado una aplicación que se ocupa de subir la información a un tópico de Kafka, manteniendo un fichero local con el último índice procesado del fichero de salida de OSSEC, para evitar posibles reenvíos de eventos ya transmitidos al sistema de mensajería. La aplicación hace uso de la librería *Java* de Kafka, versión 0.8.1.2, y asegura la entrega de los mensajes al *broker* en cuestión mediante un ACK, se puede ver la parte más representativa del desarrollo en el código 6.2.

Código 6.2: Agente Java eventos de OSSEC a Kafka

```
import kafka.javaapi.producer.Producer;
import kafka.producer.KeyedMessage;
import kafka.producer.ProducerConfig;
...
//Create Kafka Producer
Properties props = new Properties();
props.put("metadata.broker.list", args[1]);
props.put("serializer.class", "kafka.serializer.StringEncoder");
props.put("request.required.acks", "1");
ProducerConfig config = new ProducerConfig(props);
producer = new Producer<String,String>(config);
...
KeyedMessage<String, String> data = new KeyedMessage<String, String>(topicName, line);
producer.send(data);
...
```

6.2.3.1 Primer Experimento OSSEC2Kafka.

Como primer experimento, en la máquina *Ubuntu4*, se va a configurar en un único *broker* un tópico de Kafka con una única partición, para ver el rendimiento que se puede ofrecer. Se lanzará la herramienta desarrollada a tal efecto con el fichero agregado que se preparó en el apartado anterior (todos los eventos iniciales en un único fichero). La monitorización se realizará mediante la herramienta *kafka-web-console* [cla15], cuyos resultados de transferencia se muestran en la figura 6.6. Como *consumer* de datos se está usando la aplicación de consola que viene con el proyecto Apache Kafka.

Como se puede apreciar en la imagen el productor de Kafka, sin hacer uso de particiones, múltiples brokers, o escalar en número de hilos para entregar mensajes, obtiene un rendimiento de 20 eventos por segundo, con lo que cubre las necesidades seleccionadas en la ecuación 6.1. El consumidor de consola de Kafka, como se ve en la imagen realiza lecturas a bloques de eventos grandes, espaciados en el tiempo, aunque no se va a comentar la implementación ya que será sustituida en los siguientes experimentos.

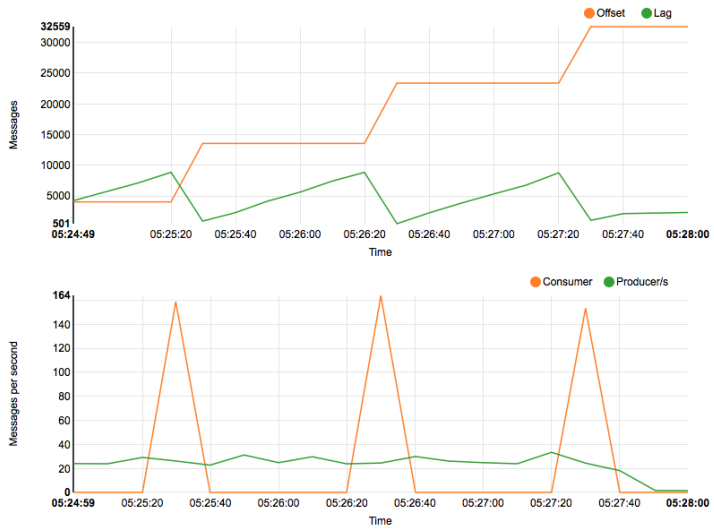


Figura 6.6: Ossec - Primer experimento con Kafka.

6.2.3.2 Segundo Experimento OSSEC2Kafka.

El segundo experimento de la capa de adquisición de OSSEC, es utilizar el agente de Kafka creado para enviar el log anterior al clúster Hadoop, haciendo uso de Spark Streaming. En este segundo experimento sólo se va a imprimir por pantalla la información recibida, como se muestra en el código 6.3. Los resultados, figura 6.7, como se puede apreciar son notablemente mejores que con el consumidor de Kafka para consola.

Código 6.3: Impresión a pantalla de Spark Streaming para eventos de Ossec

```
/*
*
* Launch:
* ./bin/spark-submit --packages org.apache.spark:spark-streaming-kafka_2.10:1.4.1 --class
    net.klobato.streaming.kafkaOssecConsumer --master spark://ubuntu1:7077 --driver-memory
    8g --executor-memory 6g /home/klobato/thesisklobato_2.10-1.0.jar ubuntu4:2181
    sparkStreasmingGroup0 ossecTest 1
*
* */
object kafkaOssecConsumer {
      def main(args:Array[String]) : Unit = {
            if(args.length < 4){
                  System.err.println("Usage: kafkaOssecConsumer <zkQuorum> <group>
                      <topics> <numThreads>")
                  System.exit(1)
            }

            val Array(zkQuorum, group, topics, numThreads) = args

            val sparkConf = new SparkConf().setAppName("KafkaOSSECConsumer")
```

```
        val ssc = new StreamingContext(sparkConf, Seconds(1))
        ssc.checkpoint("hdfs://ubuntu1:9000/user/klobato/checkpoint")

        val topicMap = topics.split(",").map((_,numThreads.toInt)).toMap
        val lines = KafkaUtils.createStream(ssc, zkQuorum, group, topicMap).map(_._2)
        lines.foreachRDD { (rdd, time) =>
                println(time)
                rdd.collect().foreach(println)
            }

        ssc.start()
        ssc.awaitTermination()
    }
}
```

Figura 6.7: Resultados Experimento 2 para eventos de Kafka a Spark Streaming con impresión de pantalla.

6.2.3.3 **Tercer Experimento OSSEC2Kafka.**

Finalmente, para dar por concluida la bondad del desarrollo, se ha creado una aplicación de Apache Spark Streaming que se ocupa de procesar el JSON que envía OSSEC con las alertas totales y tras procesarlo lo imprime en sendos ficheros de log persistidos en HDFS. Esta prueba escribe tanto los resultados originales de JSON, como el transformado a CSV, pudiendo usarse para análisis posteriores por Spark SQL, Hive y Tableau como se hará en el capítulo 7. La latencia, como se puede apreciar en la gráfica de la figura 6.8, una vez que el sistema estabiliza se sitúa en 1,419 segundos de media, lo que para el entorno del laboratorio de pruebas es más que suficiente, procesando 152,08 eventos por segundo, y teniendo en cuenta que el objetivo necesario calculado en 6.1 es notablemente inferior.

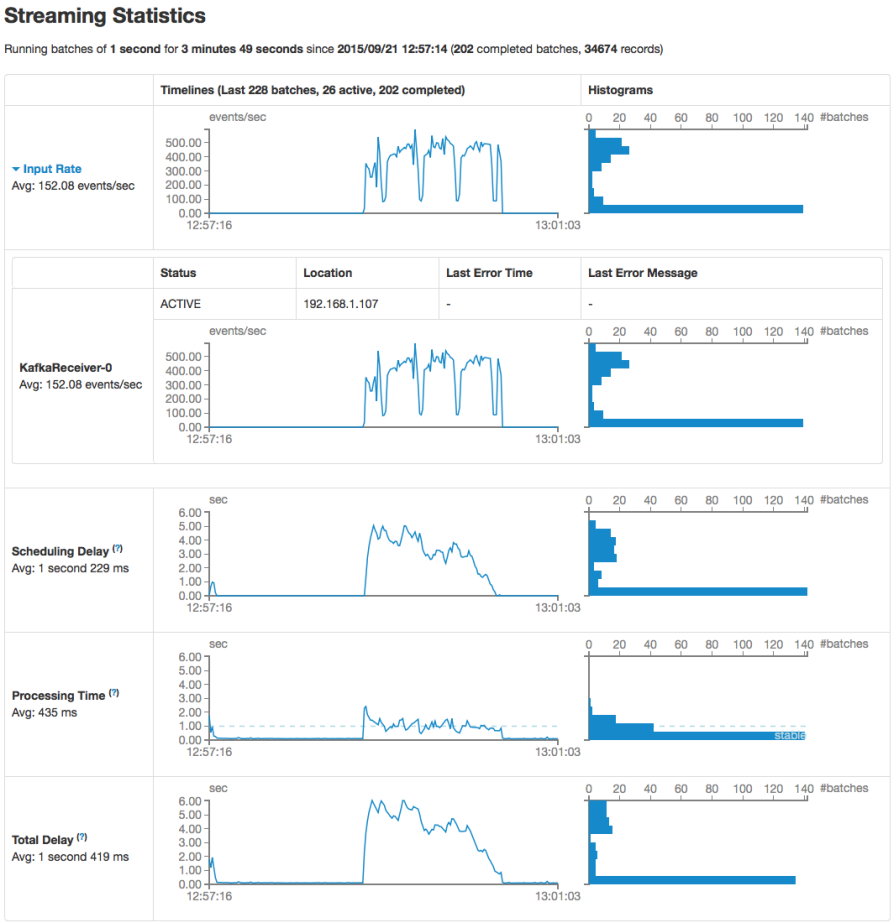


Figura 6.8: Resultados Experimento 3 para eventos de Kafka a Spark Streaming con procesado a CSV y escritura a disco.

6.3 Recogida de evidencias desde Snort.

6.3.1 Configuración de Snort con Barnyard2.

En el caso de Snort, como se indicaba en el capítulo 3 de recogida de evidencias, sección 3.4.1, generalmente para evitar la latencia del sistema en la escritura de información de paquetes de red, se suele habilitar en la configuración de Snort la salida a formato *unified2*, código 6.4. Snort se ha desplegado en la máquina *Debian-devel*, al igual que el agente central de OSSEC, y está configurado en modo *bridge* para poder recoger el tráfico generado entre los dos equipos de trabajo del laboratorio.

Código 6.4: Configuración de Snort

```
##################################################
# Step #6: Configure output plugins
# For more information, see Snort Manual, Configuring Snort - Output Modules
##################################################

# unified2
# Recommended for most installs
# output unified2: filename merged.log, limit 128, nostamp, mpls_event_types,
     vlan_event_types

output unified2: filename snort.u2

# Additional configuration for specific types of installs
# output alert_unified2: filename snort.alert, limit 128, nostamp
# output log_unified2: filename snort.log, limit 128, nostamp

# syslog
output alert_syslog: LOG_AUTH LOG_ALERT

# pcap
# output log_tcpdump: tcpdump.log

# metadata reference data. do not modify these lines
include classification.config
include reference.config
##################################################
```

6.3.2 Análisis inicial de los datos de Snort.

La información que se produce en el laboratorio se va guardando de forma continua en una base de datos MySQL, al menos de forma temporal para poder analizar la información que llega al sistema. Esta configuración se quitará en el resto de experimentos para evitar las latencias asociadas a la inserción de datos en una base de datos relacional.

En este caso, para poder visualizar la información, en vez de hacer uso de la infraestructura de Hadoop, como en el caso de OSSEC, se usa Tableau conectado directamente contra la base de datos. En concreto, los datos que se muestran a continuación son una captura de alrededor de dos horas sin ataques, en la que se han creado 3 alertas nuevas en Snort para recoger paquetes de red que de por sí no serían recogidos, código 6.5.

Código 6.5: Reglas adicionales Snort intercepción tráfico normal

```
root@debian-devel:/var/log/snort# cat /etc/snort/rules/kenneth.rules
alert icmp any any -> any any (msg: "ICMP Packet"; sid:99999; rev:1;)
alert tcp !192.168.1.200 any -> !192.168.1.200 any (msg: "TCP Packet"; sid:99998; rev:1;)
alert udp any any -> any any (msg: "UDP Packet"; sid:99997; rev:2;)
```

Estas reglas adicionales se introducen para poder enseñar a la capa de algoritmia del capítulo 7 lo que sería el tráfico normal, filtrando en la regla TCP el envío a/desde el equipo de la mensajería Kafka: *ubuntu4*. Los datos del análisis son los siguientes:

- El tamaño del archivo binario en formato *unified2* es de: 71 MB.
- Se recoge un total de 146.463 eventos.
- Se puede ver la distribución por tipo de Alerta en la figura 6.9.

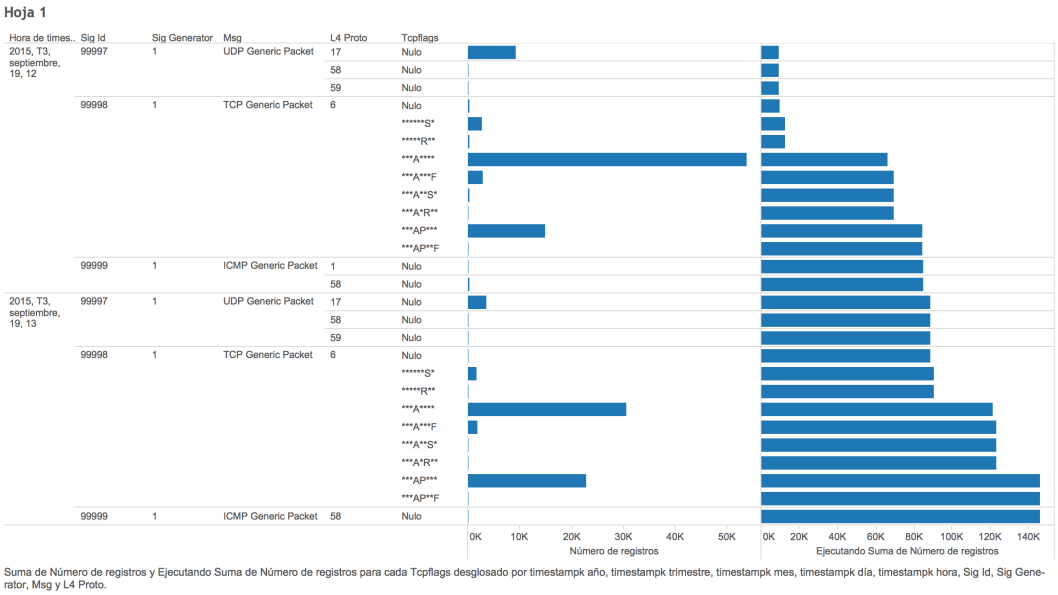


Figura 6.9: Snort - Análisis Inicial de los datos recogidos.

La estimación de generación de evidencias de esta fuente se puede ver reflejada en la ecuación 6.2.

$$146463\ eventos\ x\ \frac{1}{2h}\ x\ \frac{1h}{3600s} = 20,34\ \frac{eventos}{s} \tag{6.2}$$

Como se puede ver el número de eventos mínimos a garantizar con Snort es muy superior al requerido por OSSEC, aunque en gran medida es porque no sólo se trata de conocimiento experto con alertas concretas generadas, si no que se abre el estudio para poder analizar el tráfico clasificado como normal. Esto permite realizar la detección de anomalías y usos indebidos no basado exclusivamente en conocimiento experto a través de las conexiones de red, se verá en el capítulo 7.

6.3.3 Agente ESIDE-BIDS para Snort.

Como agente de ESIDE-BIDS se instala Barnyard2, y se configura que recoja la información y la almacene, en primera instancia en base de datos para poder analizarla más fácilmente, usada en el apartado anterior. Finalmente, para poder pasar la información a la capa de adquisición en Apache Kafka (sección 4.9), se usa una modificación de Barnyard2 [Red15]. La versión de código del repositorio anterior se ha modificado principalmente para poder compilar adecuadamente y para enviar el tamaño de los datos de los paquetes en vez de los datos en si mismos, ya que no se usan en estos experimentos. Denotar, que Barnyard2 puede funcionar en dos modos: de forma continua, en la que se manda la información constantemente; o en forma de proceso de lotes, donde se procesa el fichero que se le indique. La configuración de barnyard2 se puede ver en el código 6.6.

Código 6.6: Configuración de Barnyard2

```
...
# database: log to a variety of databases
# ----------------------------------------------------------------------------
#
# Purpose: This output module provides logging ability to a variety of databases
# See doc/README.database for additional information.
#
# Examples:
#   output database: log, mysql, user=root password=test dbname=db host=localhost
#   output database: alert, postgresql, user=snort dbname=snort
#   output database: log, odbc, user=snort dbname=snort
#   output database: log, mssql, dbname=snort user=snort password=test
#   output database: log, oracle, dbname=snort user=snort password=test
#

output database: log, mysql, user=snort password=1234 dbname=snort host=localhost
```

```
...
# Kafka Plugin configuration. Sends alerts to broker ubuntu4, topic snort
# this can be configured to use several partitions
output alert_json: kafka://ubuntu4:9092@snort
```

6.3.3.1 Primer Experimento Barnyard2Kafka.

Como primer experimento se configura en el sistema *ubuntu4* un tópico Kafka, con un único *Broker*, con una sola partición y con el consumidor de consola. Se lanza Barnyard2 en modo por lotes con el fichero de *unified2* de Snort de los datos del análisis (71MB). Se monitoriza el envío y consumo de eventos mediante la herramienta *kafka-web-console* [cla15], cuyos resultados se muestran en la figura 6.10, señalando el máximo *throughput* del sistema con esta configuración.

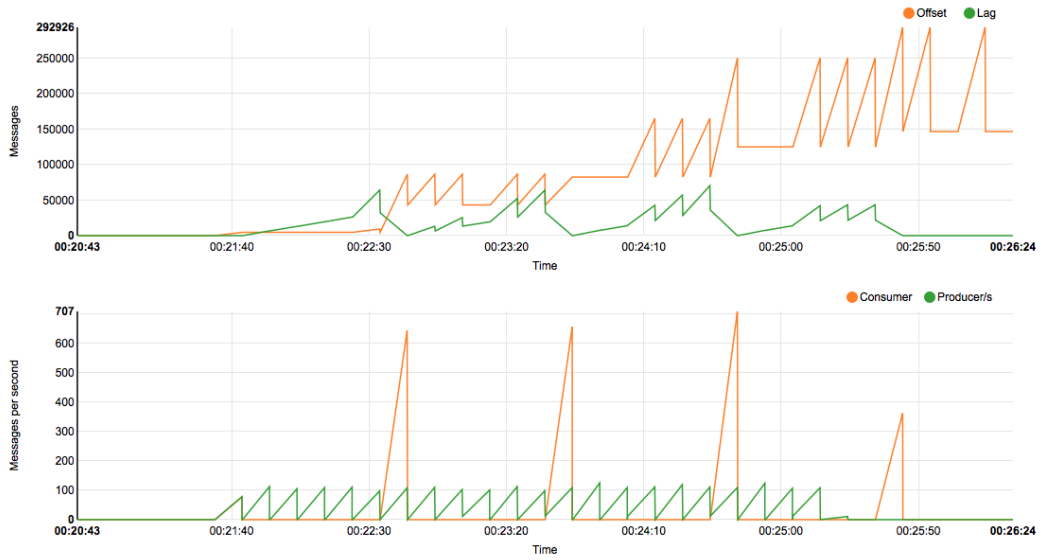


Figura 6.10: Snort - Primer experimento de Snort a Kafka.

Como se puede apreciar en la figura 6.10, la tasa de transferencia de la primera configuración ya llega a los 100 eventos por segundo, cubriendo el objetivo inicial de la ecuación 6.2.

6.3.3.2 Segundo Experimento Baryard2Kafka.

El segundo experimento de la capa de adquisición para Snort es enviar la información a Spark Streaming para realizar exclusivamente la impresión de los datos por pantalla. Los resultados se pueden ver en la figura 6.11.

Los resultados son prometedores, ya que se llega a una media de 400 eventos procesados por segundo, con una latencia media de 14 milisegundos. La configuración sigue siendo la misma, sin tener que recurrir a la escalabilidad de Apache Kafka y de Apache Spark: 1 único broker, 1 tópico, 1 partición en el tópico y sólo un ejecutor en Spark dedicado a la recogida de mensajes.

Figura 6.11: Snort - Segundo experimento de Snort a Kafka.

6.3.3.3 **Tercer Experimento Barnyard2Kafka.**

El último experimento, al igual que se hizo con OSSEC, será el envío de la información a Apache Spark, haciendo que este escriba a disco (HDFS) tanto el fichero correspondiente de JSON, como la transformación a CSV. Estos resultados serán usados como base del capítulo 7 y se pueden ver en la figura 6.12.

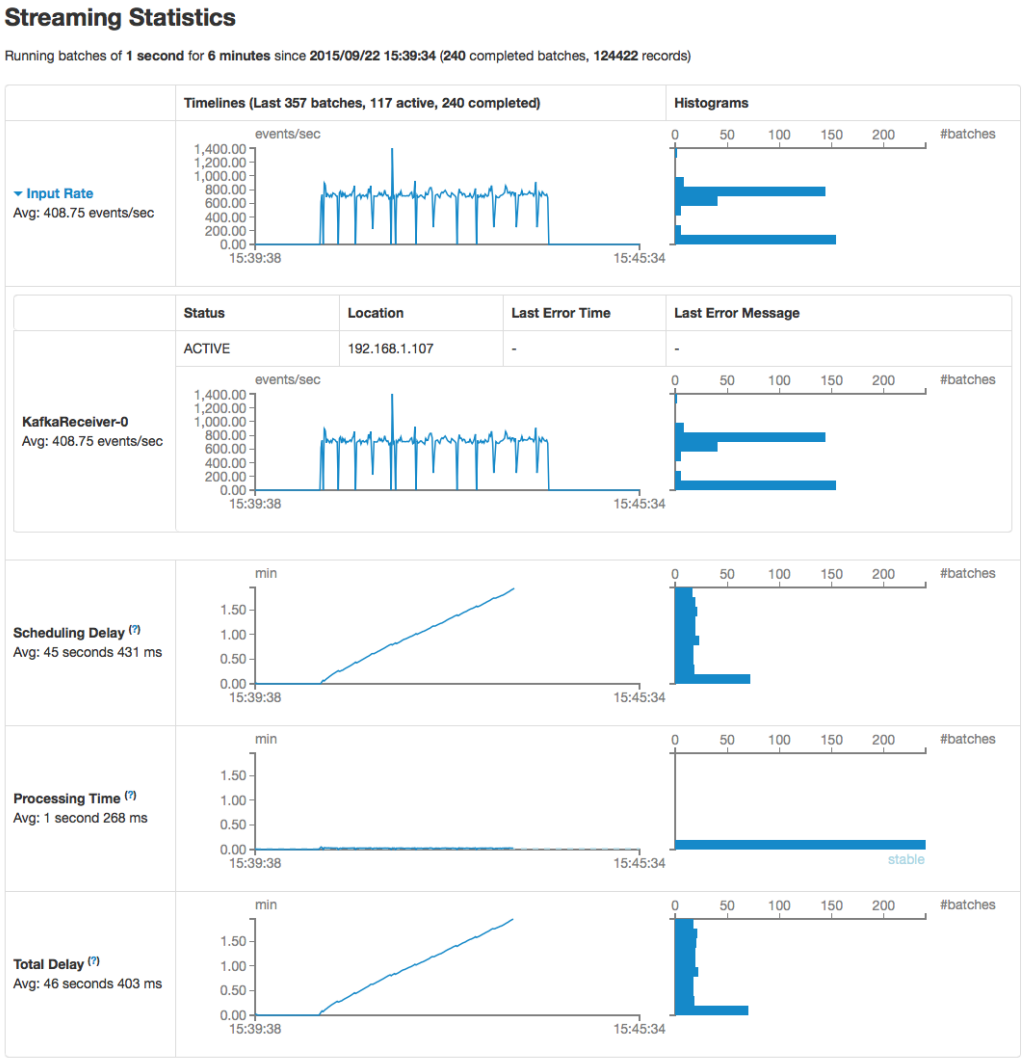


Figura 6.12: Snort - Tercer experimento de Snort a Kafka.

El experimento resume 2 horas de tráfico limpio interceptado por Snort y lo envía en menos de 5 minutos a la capa de Adquisición, llegando a pasar 1408 eventos por segundo de pico, con una media de 408 eventos

por segundo. Si bien el tiempo de procesamiento de los datos se fija en una media de 1,3 segundos, el problema dentro del laboratorio queda patente a la hora de lanzar los trabajos, debido a que no hay suficientes "ejecutores" para poder computar la información de entrada. Obviamente esta situación no es necesaria si el tráfico se recoge en tiempo real acorde a lo indicado en la ecuación 6.2, pero es una buena prueba para establecer los límites de la tecnología y el laboratorio creado.

6.4 Escalando la capa de adquisición con Apache Kafka y Apache Spark.

En las pruebas anteriores, se ha dejado la configuración por defecto y no se ha tuneado la escalabilidad del clúster en ningún momento. En la figura 6.13 se muestra un diagrama de las entidades involucradas en el proceso de transferencia de información. Los puntos en los que poder escalar son los siguientes:

- Escalabilidad de la capa de Productores.
- Escalabilidad de la capa de Apache Kafka.
- Escalabilidad en la capa de Receptores de Apache Spark.
- Escalabilidad en la capa de procesamiento.

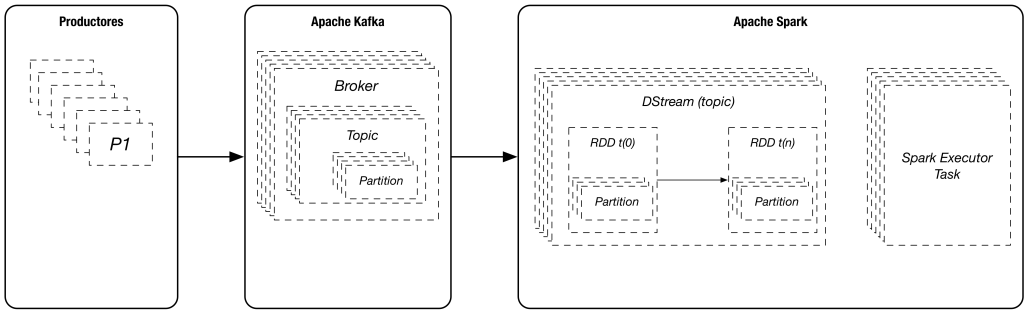


Figura 6.13: Escalabilidad Capa de Adquicisión.

La arquitectura planteada tiene numerosos puntos de ajuste para poder permitir escalar a grandes volúmenes de información. Si se tienen numeroso Productores de datos y flujos de información muy rápidos, se puede crecer en los siguientes puntos, generalmente en orden:

1. Crear el tópico con un número apropiado de particiones. El nombre del tópico al que se conectan los clientes será el mismo, pero los mensajes serán enviados a una partición diferente, que pueden encontrarse en múltiples *brokers* y que el clúster Kafka se ocupará de balancear.

2. Crear múltiples tópicos con múltiples particiones. Los clientes no sólo escalan en la partición a la que envían los resultados, si no que cambian de tópico, permitiendo incluso tener varios sistemas Kafka desplegados, repartiendo el tráfico en clústeres distintos o por lo menos, si es el mismo clúster, en máquinas diferentes.

3. Escalar el número de nodos que forman parte del clúster de Apache Kafka.

Con las mejoras anteriores lo que se consigue es tener una capa de recepción de mensajes suficientemente apropiada a la dimensión del problema a resolver.

El siguiente apartado es cómo consumir la información dentro de la arquitectura de Apache Spark. Se cumplen las siguientes reglas:

- Partiendo de un tópico con 10 particiones, si la aplicación de Apache Spark Streaming sólo se configura un hilo, las 10 particiones serán consumidas por ese hilo.

- Si en cambio se ponen 2 hijos de consumidores a trabajar, se repartirán las particiones entre ambos hilos.

- Siguiendo la línea anterior, si se tienen 10 hilos, cada consumidor procesará 1 partición.

- Si se introducen más hilos de consumo, sólo tendrán ocupación 10 de ellos, quedando el resto sin carga.

Conforme se avanza en los consumidores, si se empieza por un único hilo y se va aumentando hasta 20, generalmente se lanza un proceso en Apache Kafka que se denomina "rebalanceo", por el que cada hilo se asigna a una partición, dejando 10 hilos sin carga, pero haciendo que el hilo de partida no lea las 10 particiones si no sólo una de ellas.

Para procesar de forma paralela los mensajes, Apache Spark nos ofrece dos opciones:

- Aumentar el número de hilos que se van a desplegar en el ejecutor que se ocupa de leer los mensajes. El máximo conviene fijarlo en el número de particiones que tenga ese tópico. Esta opción tiene una pega, que todo se ejecuta en la misma máquina, haciendo que sea un cuello de botella el envío/recepción de información a través de la tarjeta de red.

- Como segunda opción, es crear varios *Streams*, tantos como particiones haya en el tópico, haciendo que los ejecutores que leen mensajes se distribuyan en el clúster en diferentes máquinas. Esta opción, haciendo uso de la clausula *union* para juntar *DStreams*, es una de las más usadas en los sistemas grandes de recogida/envío de eventos.

Hasta aquí, sólo se ha hablado de la capa de adquisición y cómo conseguir aumentar el rendimiento y escalarla convenientemente para el problema. En el caso que se ha visto en los últimos experimentos de ambas tecnologías (secciones 6.2.3.3 y 6.3.3.3 respectivamente) el problema es diferente: *No hay pega en recoger la información, pero el procesado de la misma tiene unas latencias significativas*[1].

En este punto, lo que hay que aumentar o distribuir es el procesamiento de la información y la resolución depende completamente del conjunto de máquinas que se tenga a disposición. En la figura 6.13 se muestra que los *DStreams* recibidos, están compuestos por una o varias particiones que vienen del tópico o tópicos a los que están escuchando. En Spark Streaming, hay una operación que hace que la paralelización de estos *DStreams*, cambie de número de particiones, es decir, de número de máquinas a poner a trabajar en el problema.

A modo de ejemplo demostrativo de escalabilidad, se puede ver un ejemplo del cambio para repartir la información en las 3 máquinas que conforman el cluster del laboratorio de ESIDE-BIDS, código 6.7, con la correspondiente mejora de rendimiento que se puede visualizar en la figura 6.14. En el caso de Snort, experimento 3, que es el que se ha repetido, sólo hay una partición en el tópico, con lo que se cargaría el grueso del trabajo a un único ejecutor.

[1] Denotar que los experimentos no son en tiempo real, si no inyección de eventos desde ambas fuentes para probar la capa de adquisición. Como se ha comentado, para la tasa de eventos que se producen por Snort y por OSSEC, la infraestructura puede acometerla sin latencias.

Código 6.7: Ejemplo de repartición de procesamiento en Apache Spark.

```
  ...
        val snortJsonlines = KafkaUtils.createStream(ssc, zkQuorum, group,
            topicMap).map(_._2)
/*->*/  val snortJsonlinesParallel = snortJsonlines.repartition(3)
        val snortCSVlines = snortJsonlines.map(parseSnortJson)
  ...
```

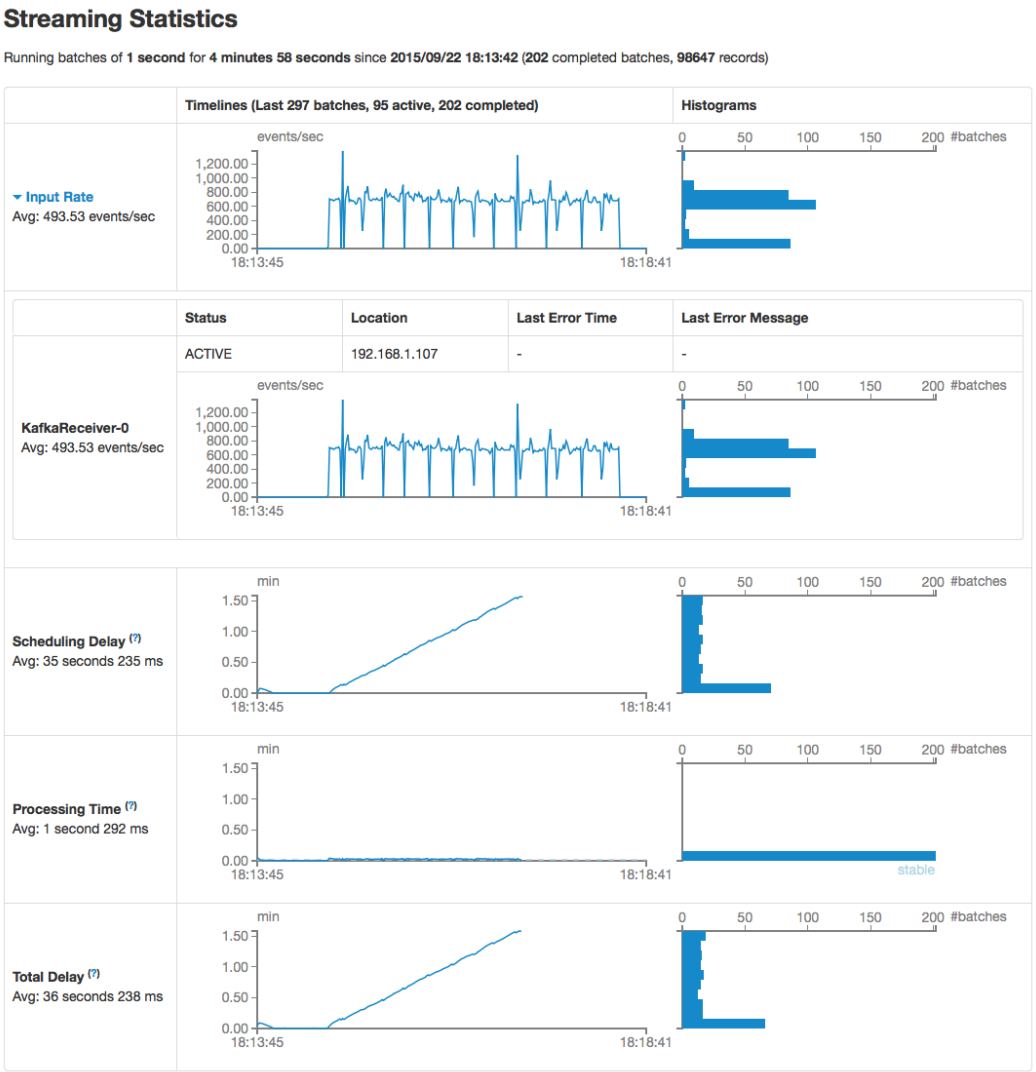


Figura 6.14: Escalado del Experimento 3 de Snort.

La mejora es notable, haciendo que se puedan procesar más eventos por segundo y reduciendo los tiempos de espera, permitiendo escalado aumentando el número de máquinas o ejecutores consumiendo de Kafka.

«Sin análisis de Big Data, las empresas están ciegas y sordas.»

Geoffrey Moore (1946)

Capítulo 7

ESIDE-BIDS - Machine Learning.

Este último capítulo de la parte de experimentación, sigue la línea de trabajo de los capítulos anteriores. En el capítulo 3 se describen las herramientas con las que se generan las evidencias tanto de *host* como de *network*. En el capítulo 4 se introducen las tecnologías que se van a usar, así como la arquitectura del nuevo modelo de ESIDE-BIDS para integrarse con los SIEMs actuales. En el primer capítulo experimental se han demostrado la capa de *machine learning* sobre un *DataSet* estático, tanto para aprendizaje supervisado como no supervisado, capítulo 5. En el capítulo 6 se trabaja a nivel experimental en la recogida de evidencias en tiempo real, haciendo uso de Spark Streaming y de las herramientas señaladas en el capítulo 3. Este capítulo viene a poner todas las piezas de trabajo juntas, realizando tanto la captura, como la experimentación de los modelos de *machine learning* a desplegar en cascada que se buscaba en la hipótesis y objetivos principales de esta tesis.

7.1 Análisis de Datos.

Para poder comprender el problema a la hora de afrontar una solución de *machine learning*, lo primero es analizar el mismo y delimitar las posibles

alternativas que se pueden ofrecer. Este primer apartado tratar de mostrar los datos tanto de máquina (*host*) como de red (*network*), con y sin ataques abordando diferentes perspectivas, tomando como base la recogida de información realizada en el capítulo 6. Para el análisis y el desarrollo se va a hacer uso de la tecnología Apache Spark [Fou15a] (Spark Core, Spark Streaming, Spark MLlib y Spark SQL) así como de Apache Hive [Fou15g], usando la herramienta Tableau [Sof15] de *Bussines Intelligence* para la inspección visual de la información.

7.1.1 **Evidencias de máquina (host).**

Como ya se indicaba en el capítulo 3, la recogida de eventos de *Host* se realiza a través de OSSEC [Mic15]. En la imagen 7.1 se puede ver la distribución del número de eventos, representados por el tamaño del cuadrado, agrupados por nivel, tipo de evento (*sid* y comentario), y localización (*Host*) donde se ha producido. Estos resultados están enlazados con la información presentada en la sección 6.2.2, ya filtrando las alertas debidas al NIDS (log de la máquina que OSSEC recogía también).

Analizando los datos durante la semana de recogida de evidencias, el mayor número de eventos proviene de la máquina *Windows Server 2008 R2* y es debido a los servicios que tiene corriendo de *SQL Server 2008 R2*, algo habitual. Los eventos en OSSEC se categorizan de menor a mayor nivel de criticidad, situando en el número 7 o por encima aquellos que deberían revisarse. En los datos normales, se observa que por encima de ese número se encuentran:

- El único evento de nivel 10 que aparece es en el equipo de escritorio que se usa para trabajar, versión Windows 10, y fue un error producido al instalar el *Device Drivers Kit* de Microsoft.
- Todos los eventos producidos de nivel 7 y 8 son debidos a la instalación y configuración del laboratorio, así como cambios en permisos y privilegios que se han debido de realizar para poder desplegar todo el software.

Tras lanzar la herramienta Nessus [tns15], cuyos resultados se encuentra en la figura 7.2, se han generado nuevas evidencias no contempladas en los eventos mostrados en la figura 7.1. Para facilitar la labor del lector, se filtran todos aquellos eventos no existentes en la captura de tráfico normal.

Como se puede apreciar en la figura 7.2,la herramienta Nessus realiza varios ataques adicionales en diferentes niveles:

- Como era de esperar, la imagen más atacada ha sido *Metasploitable*.
- El mayor número de eventos ha sido de ataques Web, incluyendo ataques XSS.
- Se trata de vencer a fuerza bruta varios servicios, generando intentos de acceso no satisfactorios.
- Se consigue explotar servicios Web de forma satisfactoria.
- Se trata de acceder a las máquinas del clúster por SSH de forma insegura intentando ganar una cuenta de acceso.

Siguiendo con las herramienta de *Pentesting*, para continuar con la recogida de evidencias y el análisis a nivel de *Host* se lanza *Metasploit Framework*, primero como escáner y después en modo explotación automático de las vulnerabilidades localizadas. El resumen de los ataques identificados, filtrando las evidencias que ya aparecían en la captura normal, aparece reflejado en la figura 7.3.

Comparando los resultados entre Nessus y Metasploit, ambos localizan los servicios de FTP y tratan de vulnerarlos, así como los servicios de DNS, y realizan su mayor trabajo en atacar servidores Web. Metasploit adicionalmente localiza un servidor de SQL Server y trata de logearse por fuerza bruta, así como lanzar varias *SQL Injections*. Ambas herramientas encuentran un antiguo servicio de login remoto (RSH) y realizan ataques XSS (*Cross Site Scripting*). Metasploit llega a ofrecer una *shell* remota explotando una vulnerabilidad en el servidor de Samba, valiéndose de una *shellcode* genérica de *reverse shell*[1].

[1] Se conecta como cliente del equipo que ataca

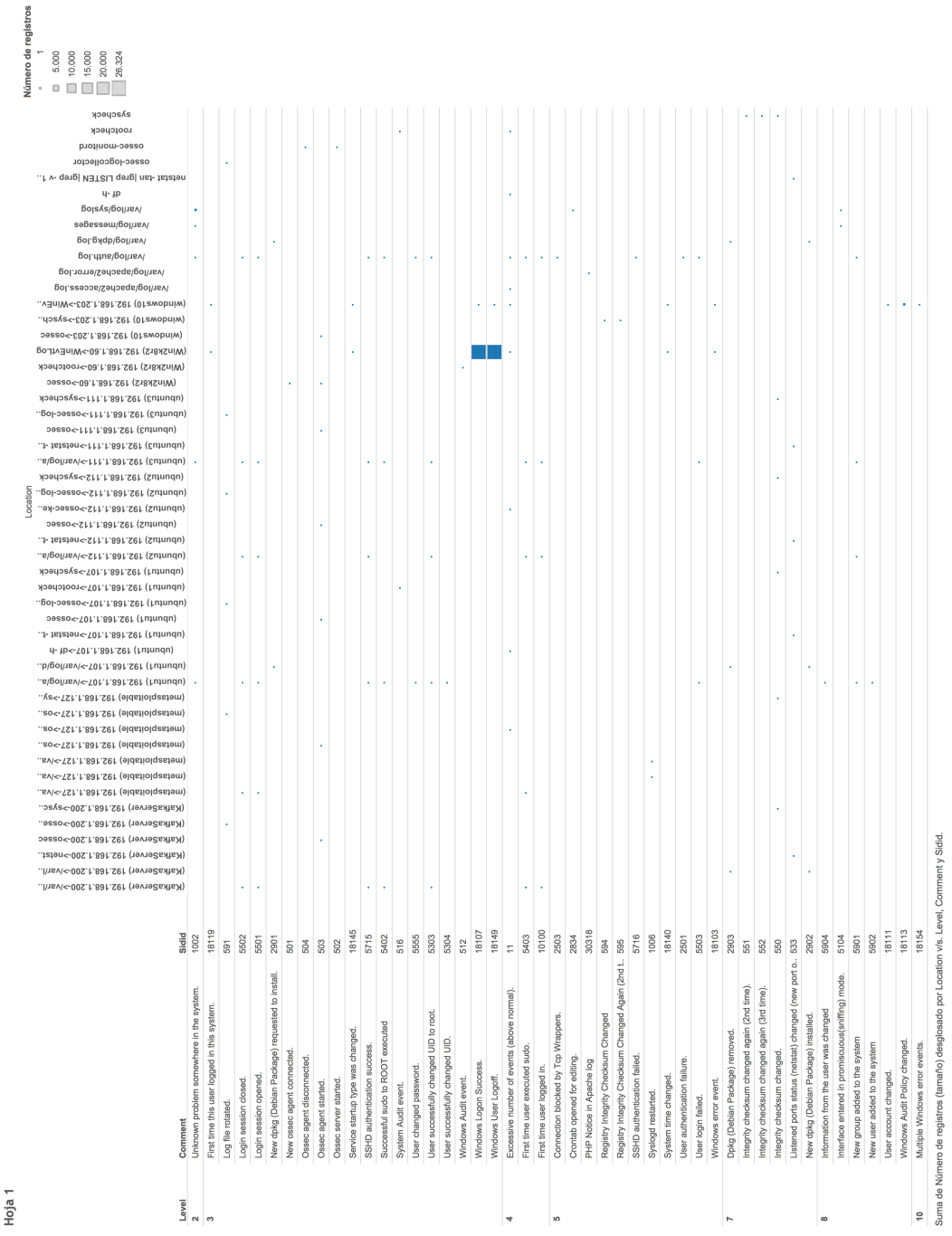
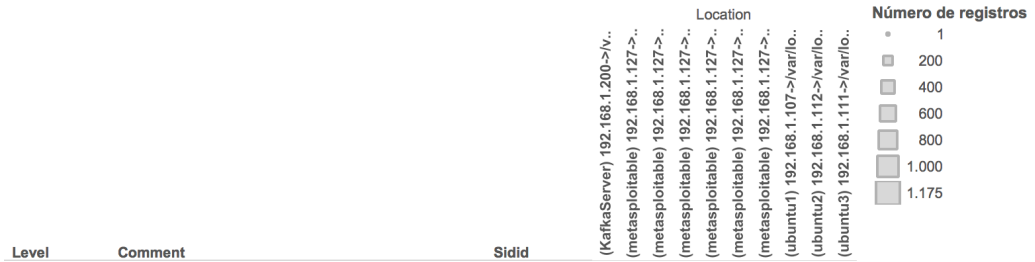


Figura 7.1: Eventos recogidos de Host sin ataques.



Level	Comment	Sidid
3	Bad zone transfer request.	12139
	FTP Authentication success.	11402
	FTP session opened.	11201
		11401
	Remote host disconnected due to login time out.	11215
4	Couldn't open the incoming connection. Check log mes..	11222
5	Attempt to access forbidden file or directory.	30105
	Attempt to login using a non-existent user.	11203
	Invalid URI (bad client request).	30115
	Login failed accessing the FTP server	11204
	Login failed accessing the FTP server.	11403
	Web server 400 error code.	31101
6	A web attack returned code 200 (success).	31106
	Code Red attack.	30107
	Common web attack.	31104
	SSH insecure connection attempt (scan).	5706
	Suspicious URL access.	31516
	XSS (Cross Site Scripting) attempt.	31105
10	Connection to rshd from unprivileged port. Possible net..	2551
	Multiple common web attacks from same source ip.	31153
	Multiple connection attempts from same source.	11252
	Multiple failed login attempts.	11210
	Multiple FTP connection attempts from same source IP.	11452
	Multiple web server 400 error codes from same source i..	31151
	Multiple XSS (Cross Site Scripting) attempts from same..	31154

Figura 7.2: Eventos adicionales recogidos de Host tras lanzar Nessus.

Level	Comment	Sidid
3	Bad zone transfer request.	12139
	FTP Authentication success.	11402
	FTP session opened.	11201
		11401
	Physical root login.	2505
	Remote host disconnected due to login time out.	11215
4	Excessive number of events (above normal).	11
5	Attempt to access forbidden file or directory.	30105
	Attempt to login using a non-existent user.	11203
	Invalid URI (bad client request).	30115
		30315
	Login failed accessing the FTP server	11204
	Login failed accessing the FTP server.	11403
	Logon Failure - Unknown user or bad password.	18130
	MS SQL Server Logon Failure.	18180
	Web server 400 error code.	31101
6	A web attack returned code 200 (success).	31106
	Code Red attack.	30107
	Common web attack.	31104
	osCommerce file manager login.php bypass attempt.	31504
	PHPMyAdmin scans (looking for setup.php).	31515
	SQL injection attempt.	31103
		31164
	SSH insecure connection attempt (scan).	5706
	Suspicious URL access.	31516
	XSS (Cross Site Scripting) attempt.	31105
10	Connection to rshd from unprivileged port. Possible network scan.	2551
	Multiple common web attacks from same source ip.	31153
	Multiple connection attempts from same source.	11252
	Multiple failed login attempts.	11210
	Multiple FTP connection attempts from same source IP.	11452
	Multiple web server 400 error codes from same source ip.	31151
	Multiple XSS (Cross Site Scripting) attempts from same source ip.	31154
13	Non standard syslog message (size too large).	1003

Figura 7.3: Eventos adicionales recogidos de Host tras lanzar Metasploit Framework.

7.1.2 **Evidencias de red.**

En el capítulo 3 se indicaba que la recogida de eventos de red se realiza a través de Snort [aiA15]. Los experimentos de recogida de información así como la configuración y reglas del mismo se adelantaban en la sección 6.3. En resumen:

- Se han introducido tres reglas genéricas para capturar todos los paquetes de TCP, UDP e ICMP. Esto es debido a que no sólo se busca conocimiento experto, si no eventos que puedan ser detectados como anomalías, y para ello hacen falta más datos que exclusivamente las alertas proporcionadas por el conocimiento experto que están reflejadas en las reglas de ambas tecnologías (host y red).
- En el campo *payload*, dado que en estos experimentos de clusterización o clasificación no se va a hacer uso de los datos, se indica el tamaño del mismo, obviando el contenido.
- Los resultados iniciales de la prueba sin ataques de 2 horas de duración se pueden ver en la figura 7.4, que como ya se introdujo son de otro orden de escala en comparación a los eventos generados por OSSEC.

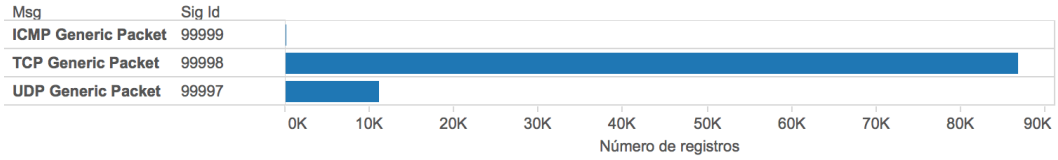


Figura 7.4: Eventos genéricos TCP, UDP e ICMP recogidos por Snort.

El total de las alertas generadas por Snort para el tráfico normal sólo activa las reglas esperadas, es decir las 3 reglas genéricas de recogida de paquetes que no han coincidido con ninguna otra regla que Snort tenga configurada. A este respecto, indicar que se han activado todas las reglas de Snort que se pueden descargar con *Suscripción* desde su página web. Se debe comentar, como dato importante, el hecho de que Snort sólo va a pasar una alerta por paquete de datos de red. En caso de que varias alertas puedan detectar el mismo paquete, se aplica siempre la más restrictiva, es decir, la que sea menos general. Las reglas adicionales que se pusieron a Snort son lo más genéricas posible, así que en caso de que una regla señale un ataque concreto, siempre saltará esa frente a las genéricas que sirven para recoger

evidencias adicionales. Esto evita que se tengan paquetes duplicados con dos alertas o más diferentes.

El grueso de los ataques de Nessus, figura 7.5, detectados por las reglas de Snort se categorizan en indicadores de escaneos de red, detección de alguna de las herramientas de Nessus que tratan de explotar *vulnerabilidades*, y diversos mecanismos para recabar información de los equipos en la red: paquetes ICMP formateados con opciones especiales, tráfico SNMP, etc. La captura de Snort en formato *unified2* tiene un tamaño de 43 MB.

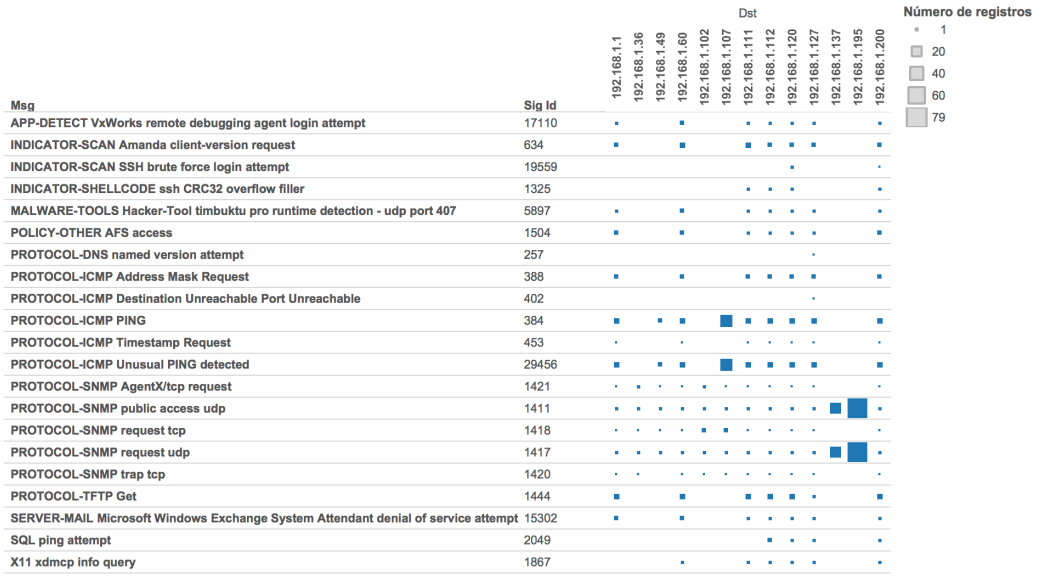


Figura 7.5: Alertas de Snort tras lanzar Nessus sobre el laboratorio E-BIDS.

Se repite el proceso realizado con Nessus, pero esta vez con Metasploit [Rap15]. La captura dura bastante más tiempo, en concreto 3 horas y 16 minutos, generando 3 ficheros en formato *unified2* de Snort (dos de 128 MB y uno de 74MB), que sigue con todas las reglas activas. Los resultados se pueden ver reflejados en la figura 7.6.

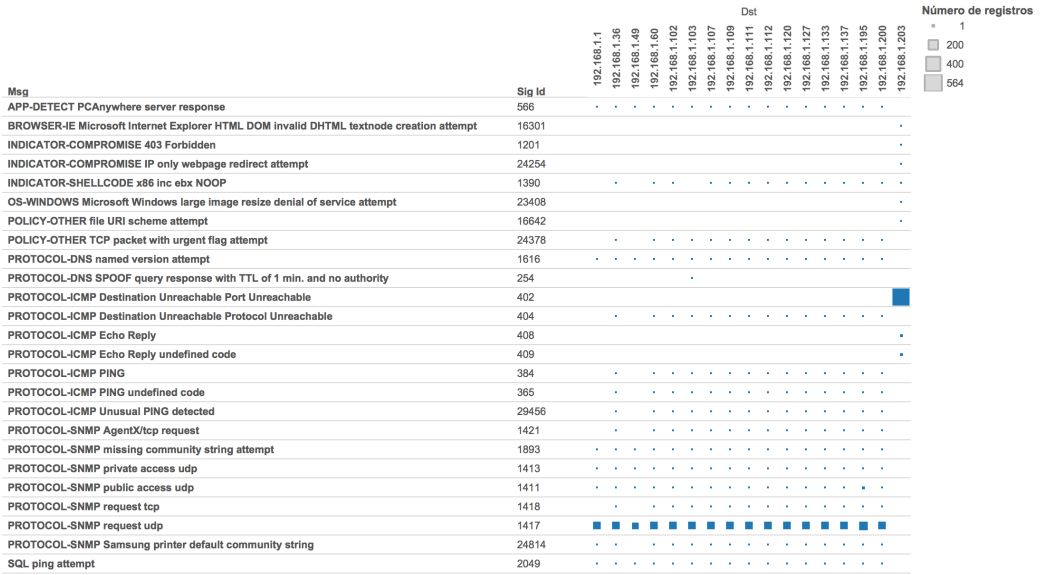


Figura 7.6: Alertas de Snort tras lanzar Metasploit sobre el laboratorio E-BIDS.

Como se puede apreciar en las imágenes 7.5 y 7.6 de Nessus y de Metasploit respectivamente, ambas herramientas levantan una serie de alertas de Snort relativas a la inspección de host remota. Además en ambos casos se identifican vectores de ataque contra servicios tratando de explotar vulnerabilidades.

Una vez realizada la captura de evidencias, el siguiente paso es la creación de modelos que puedan tratar el tipo de vectores de evidencias que llegan a través de la capa de recolección, así como estimar los parámetros más apropiados para todos ellos. Los siguientes puntos parten del trabajo realizado en el capítulo 5, donde se explicaban los principales algoritmos, técnicas de aprendizaje supervisado y no supervisado, así como diferentes métricas para evaluación de la bondad de los modelos.

7.2 Clusterización.

El proceso de clusterización ya fue explicado en detalle en la sección 5.2, y en este análisis de la información se va a seguir el mismo proceso de preparación y normalización de los datos, así como el mismo algoritmo: *K-means*. El proceso a seguir se resume en la figura 7.7.

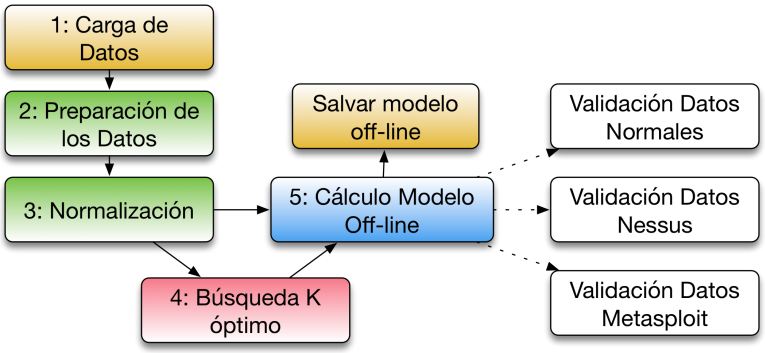


Figura 7.7: Proceso de clusterización.

7.2.1 Eventos de Host

De todas las evidencias recibidas de OSSEC, reflejadas en la *case class Ossec* del código 7.1, sólo se va a tener en cuenta el SIDID, identificativo único de la evidencia generada en el host, y la localización de la misma, es decir, el equipo en el que se ha producido, usando *level* como el nivel de criticidad de cada evidencia, aunque no forma parte del modelo. Con ello se generan de forma análoga al proceso de la sección 5.1.1 las evidencias en forma de vectores de características (0.0, 0.0, 1.0, ...), siendo sendas etiquetas normalizadas.

Código 7.1: Case class representando evidencia de Ossec

```
case class Ossec (sidid: String,
       level: String, comment: String,
       info: String, dstuser:String,
       srcuser: String, location:String,
       srcip: String, fulllog: String,
       md5_after: String, md5_before: String,
       path: String)
```

Sobre el conjunto de datos etiquetado como sin ataques, se realiza una búsqueda de los parámetros óptimos del modelo K-means al igual que en la sección 5.2.1, obteniendo los resultados de la tabla 7.1

Cuadro 7.1: Datos crudos cálculo K-Means óptimo para OSSEC.

K	Dist. Media Centroide	Entropía Etiquetas
20	1.6732099382361991	0.14610655322247232
30	1.3317336462554652	0.0761367544252166
40	1.062299258908871	0.049516281768724905
50	0.7674740701643552	0.030443640091635957
60	0.5686818546121224	0.015508132301465508
70	0.5312084306563175	0.019566578860880725
80	0.26546094016293	0.001738566415408632
90	0.10286353772724809	5.806525261489932E-4
100	0.022179131234720612	1.8159408051550082E-4
110	0.009240168922998643	1.8159408051550082E-4
120	0.0028491465442234535	1.8111584555238958E-4
130	1.7332667490251246E-15	0.0
140	2.624493269952116E-15	0.0
150	2.5866809720422046E-15	0.0
160	3.3291255947625017E-15	0.0
170	1.6513228982674264E-15	0.0
180	3.4282700326297516E-16	0.0
190	1.2845480384116454E-15	0.0
200	2.412463686896625E-15	0.0

El parámetro seleccionado para K es $K = 150$, teniendo una entropía excelente a nivel de etiquetas diferentes en el cluster. Para verificar la bondad modelo calculado, se procede a realizar el análisis de los *datasets* normal, nessus y metasploit, midiendo para cada evidencia tanto la distancia media con el centroide en el grupo, como entropía en los tipos de etiquetas que se tienen en el grupo. Los resultados de Nessus y Metasploit se tienen en las figuras 7.8) y 7.9. Se puede verificar que la distancia de las evidencias ambos datasets, Nessus y Metasploit, con respecto a los grupos en los que se clasifica es en general notablemente superior a la máxima distancia del grupo en comparación con el *dataset* normal, luego serían una **anomalía** para el modelo seleccionado.

label (Nessus)	cluster (Nessus)	Máx. Distance	Máx. distance (Nessus)	Mín. Distance	Mín. distance (Nessus)	Prom. Distance	Prom. distance (Nessus)
3	27	0,00	46,70	0,00	0,97	0,00	1,37
	32	0,00	11,11	0,00	11,11	0,00	11,11
	130	0,00	0,95	0,00	0,95	0,00	0,95
	135	0,00	22,71	0,00	12,18	0,00	16,40
	142	0,00	12,80	0,00	12,80	0,00	12,80
	146	0,00	10,53	0,00	10,53	0,00	10,53
5	2	0,00	34,92	0,00	34,92	0,00	34,92
	27	0,00	54,64	0,00	2,93	0,00	4,60
6	27	0,00	77,07	0,00	11,15	0,00	14,91
7	66	0,00	20,67	0,00	20,67	0,00	20,67
	80	0,00	11,55	0,00	11,55	0,00	11,55
	85	0,00	9,34	0,00	9,34	0,00	9,34
	109	0,00	8,14	0,00	8,14	0,00	8,14
10	27	0,00	76,70	0,00	6,65	0,00	10,25

Figura 7.8: Resultados clusterización K-means de evidencias de Nessus vs Dataset Normal.

label (Metasploit)	cluster (Metasploit)	Máx. Distance	Máx. distance (Metasploit)	Mín. Distance	Mín. distance (Metasploit)	Prom. Distance	Prom. distance (Metasploit)
3	27	0,00	1,64	0,00	1,64	0,00	1,64
	32	0,00	19,95	0,00	6,71	0,00	15,54
	75	0,00	43,33	0,00	43,33	0,00	43,33
	78	0,00	36,91	0,00	36,91	0,00	36,91
	130	0,00	67,05	0,00	1,61	0,00	3,14
	142	0,00	10,89	0,00	10,89	0,00	10,89
	146	0,00	7,37	0,00	7,37	0,00	7,37
4	130	0,00	58,81	0,00	58,81	0,00	58,81
5	130	0,00	67,06	0,00	2,66	0,00	3,89
	137	0,00	86,18	0,00	37,32	0,00	61,75
6	98	0,00	18,54	0,00	18,54	0,00	18,54
	130	0,00	65,23	0,00	9,23	0,00	11,29
	146	0,00	18,50	0,00	18,50	0,00	18,50
7	66	0,00	34,55	0,00	34,55	0,00	34,55
	80	0,00	52,73	0,00	52,73	0,00	52,73
	85	0,00	44,11	0,00	44,11	0,00	44,11
	109	0,00	35,43	0,00	35,43	0,00	35,43
10	130	0,00	67,05	0,00	5,76	0,00	10,51

Figura 7.9: Resultados clusterización K-means de evidencias de Metasploit vs Dataset Normal.

7.2.2 **Eventos de Red.**

De todas las evidencias recibidas de Snort, reflejadas en la *case class* del código 7.2, hay que tener en cuenta que si bien exhibe varios campos numéricos, el grueso son campos de características, por lo que el proceso de adaptación va a ser más complicado que en el punto anterior. La tabla 7.2 resume cada campo, el tipo y si se incluye o no en el modelo, junto con la explicación. Con ello se generan de forma análoga al proceso de la sección 5.1.1 las evidencias en forma de vectores de características (0.0, 0.0, 1.0, ...), para después poder ser normalizadas convenientemente.

Código 7.2: Case class representando evidencia de Snort

```
case class Snort (timestamp: String,l4_proto: String,sig_generator: String, sig_id: String,
       rev: String, priority: String,classification: String, action: String ,
       msg: String, payloadLength: String, src: String, dst: String,
       src_port: String, dst_port: String, ethlength: String,
       tcpflags: String, tcplen: String, tcpwindow: String, ttl: String, tos: String,
       iplen: String, icmpid: String, icmpcode: String , icmpseq: String, icmptype: String)
```

Sobre el conjunto de datos etiquetado como sin ataques, se realiza una búsqueda de los parámetros óptimos del modelo K-means al igual que en la sección 5.2.1, obteniendo los resultados de la tabla 7.3

El parámetro seleccionado para K es $K = 260$, ya que presenta una buena distancia media de los elementos, así como una entropía muy aceptable, con un coste computacional adecuado. Igual que para el caso de evidencias de host, se pasan los *datasets* normal, Nessus y Metasploit por el modelo para verificar la bondad del mismo. Los resultados de Nessus y Metasploit se tienen en las figuras 7.10) y 7.11, donde se muestran los mismo en tablas exportadas desde el software Tableau. En los resultados mostrados, se han filtrado las etiquetas genéricas, y se presentan la etiqueta de grupo para Metasploit, el cluster y las medidas comparativas entre normal y ataque con Nessus o Metasploit. Se puede apreciar que la detección de **anomalías** sería sencillo, con respecto al tráfico aprendido en la generación del modelo, este punto se verá más adelante.

Cuadro 7.2: Campos de evidencias de Snort.

Campo	Tipo	Incluido	Explicación
timestamp	-	No	No es una característica para el modelo.
l4proto	Feature	Sí	Tipo de protocolo L4.
sig-generator	-	No	Información adicional de la regla activa.
sig-id	Label	No	
rev	-	No	Revisión de la regla activa.
priority	-	No	Prioridad regla activa.
classification	-	No	Clasificación regla activa.
action	-	No	Acción regla activa.
msg	Label	Sí	Unido a sig-id para tener la etiqueta.
payloadLength	Numeric	Sí	Tamaño del paquete de datos.
src	Feature	Sí	Origen del paquete.
dst	Feature	Sí	Destino del paquete.
src-port	Numeric	Sí	Puerto origen (si lo hubiera)
dst-port	Numeric	Sí	Puerto destino (silo hubiera)
ethLength	Numeric	Sí	Tamaño cabecera Ethernet.
tcpFlags	Feature	Sí	Flags protocolo TCP.
tcpLen	Numeric	Sí	Tamaño cabecera TCP.
tcpWindow	Numeric	No	Muy variable.
ttl	Numeric	Sí	Tiempo de vida.
tos	Feature	Sí	Flags Tipo de servicio
iplen	Numeric	Sí	Tamaño paquete IP.
icmpid	Feature	No	Muy variable.
icmpcode	Feature	Sí	Campo CODE de ICMP.
icmpseq	Numeric	No	Muy variable.
icmptype	Feature	Sí	Campo TYPE de ICMP.

Cuadro 7.3: Datos crudos cálculo K-Means óptimo para Snort.

K	Dist. Media Centroide	Entropía Etiquetas
30	7.999113165373547	0.024181436029120592
40	7.905271855671697	0.02514120628636149
50	7.824907364235682	0.02696194089871189
60	7.975249184262351	0.15064101834472357
70	7.767100605433507	0.0202315670910921
80	7.03116471147948	0.1197590194250243
90	7.1147490726880696	0.009969114302199588
100	7.146949777489741	0.008923456094391834
110	5.997284040777677	0.014934927667949164
120	5.984560629561922	0.010474193993638662
130	5.323103555898541	0.010439611477602131
140	5.085615062345128	0.013368788734152065
150	4.984542777092605	0.015286226397712805
160	5.133343356157283	0.010196834494131187
170	4.133415778634509	0.01729006876539215
180	4.417977928105326	0.012403203228845956
190	3.5274667084329763	0.010546926873676641
200	3.817718403345419	0.019346962384083004
210	3.812636198180798	0.008777193493231725
220	3.0403154040653604	0.009337080415454307
230	3.2413210026978336	0.012993138874473108
240	2.918157226945297	0.006850293751661879
250	2.2713861059871574	0.010741334115595696
260	1.9704483323786108	0.012512026737380768
270	2.004317834938073	0.015957522775692665
280	1.43232103842112	0.006058092364321668
290	1.7070597530791287	0.010506911138660117
300	1.2967490982514747	0.006550162973223631

label (snortNessus)	Cluster	Máx. Distance	Máx. distance (snortNessus)	Mín. Distance	Mín. distance (snortNessus)	Prom. Distance	Prom. distance (snortNessus)
257 PROTOCOL-DNS named version attempt	77	195,9	8,1	0,9	8,1	2,7	8,1
384 PROTOCOL-ICMP PING	164	97,9	41,4	1,7	35,0	3,4	36,0
388 PROTOCOL-ICMP Address Mask Request	164	97,9	59,8	1,7	56,4	3,4	56,9
402 PROTOCOL-ICMP Destination Unreachable Port Unreachable	14	33,2	237,4	23,7	237,4	27,7	237,4
453 PROTOCOL-ICMP Timestamp Request	164	97,9	90,3	1,7	88,0	3,4	88,4
634 INDICATOR-SCAN Amanda client-version request	164	97,9	21,5	1,7	7,5	3,4	9,9
1325 INDICATOR-SHELLCODE ssh CRC32 overflow filler	77	195,9	21,3	0,9	7,1	2,7	10,9
1411 PROTOCOL-SNMP public access udp	164	97,9	21,3	1,7	7,2	3,4	11,0
	204	18,6	12,7	1,8	12,6	2,8	12,6
1417 PROTOCOL-SNMP request udp	164	97,9	21,3	1,7	7,2	3,4	11,0
	204	18,6	12,7	1,8	12,6	2,8	12,6
1418 PROTOCOL-SNMP request tcp	34	113,1	4,5	1,0	4,3	3,9	4,4
	77	195,9	20,6	0,9	3,6	2,7	6,9
1420 PROTOCOL-SNMP trap tcp	34	113,1	4,9	1,0	4,9	3,9	4,9
	77	195,9	20,4	0,9	3,6	2,7	6,6
1421 PROTOCOL-SNMP AgentX/tcp request	34	113,1	5,3	1,0	4,6	3,9	4,9
	77	195,9	20,5	0,9	3,6	2,7	6,9
1444 PROTOCOL-TFTP Get	164	97,9	21,5	1,7	7,1	3,4	9,9
1504 POLICY-OTHER AFS access	164	97,9	21,3	1,7	7,2	3,4	10,2
1867 X11 xdmcp info query	164	97,9	21,3	1,7	7,0	3,4	9,9
2049 SQL ping attempt	164	97,9	21,3	1,7	7,1	3,4	10,6
5897 MALWARE-TOOLS Hacker-Tool timbuktu pro runtime detection - udp..	164	97,9	21,3	1,7	7,1	3,4	9,5
15302 SERVER-MAIL Microsoft Windows Exchange System Attenda..	164	97,9	21,3	1,7	7,0	3,4	9,4
17110 APP-DETECT VxWorks remote debugging agent login attempt	164	97,9	21,5	1,7	7,7	3,4	10,0
19559 INDICATOR-SCAN SSH brute force login attempt	77	195,9	21,3	0,9	8,1	2,7	12,5
29456 PROTOCOL-ICMP Unusual PING detected	164	97,9	41,4	1,7	35,0	3,4	36,0

Figura 7.10: Resultados clusterización K-means de evidencias de Dataset Normal vs Nessus.

label (snortMeta)	Cluster	Máx. Distance	Máx. distance (snortMeta)	Mín. Distance	Mín. distance (snortMeta)	Prom. Distance	Prom. distance (snortMeta)
254 PROTOCOL-DNS SPOOF query response with T..	211	5,5	28,2	2,7	28,2	3,2	28,2
365 PROTOCOL-ICMP PING undefined code	34	113,1	54,1	1,0	54,1	3,9	54,1
	164	97,9	56,6	1,7	53,8	3,4	54,5
384 PROTOCOL-ICMP PING	31	3,2	56,0	0,0	53,2	0,0	53,9
402 PROTOCOL-ICMP Destination Unreachable Port Unreachable	0	4,8	31,7	0,0	31,7	0,2	31,7
	2	169,8	23,2	1,1	23,2	3,0	23,2
	31	3,2	60,1	0,0	31,2	0,0	37,9
	142	19,8	27,4	6,2	27,4	9,5	27,4
	156	0,8	19,5	0,1	19,5	0,2	19,5
	164	97,9	26,9	1,7	20,3	3,4	21,5
	226	1,6	38,4	0,0	38,4	0,3	38,4
	232	23,9	23,8	2,9	23,8	5,9	23,8
404 PROTOCOL-ICMP Destination Unreachable Protocol Unreachable	34	113,1	42,2	1,0	42,2	3,9	42,2
	164	97,9	45,3	1,7	41,9	3,4	42,7
408 PROTOCOL-ICMP Echo Reply	0	4,8	40,4	0,0	40,4	0,2	40,4
	31	3,2	44,1	0,0	40,0	0,0	41,5
	156	0,8	9,8	0,1	9,8	0,2	9,8
	164	97,9	11,3	1,7	11,3	3,4	11,3
	226	1,6	45,9	0,0	45,9	0,3	45,9
	232	23,9	16,8	2,9	16,8	5,9	16,8
409 PROTOCOL-ICMP Echo Reply undefined code	2	169,8	42,7	1,1	42,7	3,0	42,7
	67	1,6	42,1	0,0	42,1	0,3	42,1
	164	97,9	44,9	1,7	40,9	3,4	41,8
	184	1,3	42,1	0,0	42,1	0,3	42,1
	226	1,6	45,7	0,0	45,7	0,3	45,7
	232	23,9	43,0	2,9	43,0	5,9	43,0
566 APP-DETECT PCAnywhere server response	34	113,1	8,1	1,0	8,1	3,9	8,1
	164	97,9	18,5	1,7	4,0	3,4	9,5
	259	5,7	15,1	3,1	15,1	3,7	15,1
1201 INDICATOR-COMPROMISE 403 Forbidden	77	195,9	9,8	0,9	9,8	2,7	9,8
1390 INDICATOR-SHELLCODE x86 inc ebx NOOP	0	4,8	31,7	0,0	31,7	0,2	31,7
	31	3,2	33,7	0,0	31,2	0,0	32,4
	34	113,1	8,4	1,0	8,4	3,9	8,4
	156	0,8	19,5	0,1	19,5	0,2	19,5
	164	97,9	17,9	1,7	4,2	3,4	9,0
	226	1,6	38,4	0,0	38,4	0,3	38,4
1411 PROTOCOL-SNMP public access udp	34	113,1	8,1	1,0	8,1	3,9	8,1
	164	97,9	18,5	1,7	4,0	3,4	9,3
	259	5,7	15,1	3,1	15,1	3,7	15,1
1413 PROTOCOL-SNMP private access udp	34	113,1	8,1	1,0	8,1	3,9	8,1
	164	97,9	18,5	1,7	4,0	3,4	9,5
	259	5,7	15,1	3,1	15,1	3,7	15,1
1417 PROTOCOL-SNMP request udp	34	113,1	8,1	1,0	8,1	3,9	8,1
	164	97,9	18,5	1,7	4,0	3,4	9,5
	259	5,7	15,1	3,1	15,1	3,7	15,1
1418 PROTOCOL-SNMP request tcp	3	113,2	18,0	0,1	4,7	0,7	9,3
	34	113,1	6,2	1,0	6,1	3,9	6,2
1421 PROTOCOL-SNMP AgentX/tcp request	3	113,2	18,0	0,1	4,7	0,7	9,3
	34	113,1	6,3	1,0	6,1	3,9	6,2
1616 PROTOCOL-DNS named version attempt	34	113,1	8,1	1,0	8,1	3,9	8,1
	164	97,9	18,5	1,7	4,0	3,4	9,5
	259	5,7	15,1	3,1	15,1	3,7	15,1
1893 PROTOCOL-SNMP missing community string attempt	34	113,1	8,1	1,0	8,1	3,9	8,1
	164	97,9	18,5	1,7	4,0	3,4	9,5
	259	5,7	15,1	3,1	15,1	3,7	15,1
2049 SQL ping attempt	34	113,1	8,1	1,0	8,1	3,9	8,1
	164	97,9	18,5	1,7	4,0	3,4	9,5
	259	5,7	15,1	3,1	15,1	3,7	15,1
16301 BROWSER-IE Microsoft Internet Explorer HT..	2	169,8	4,2	1,1	4,2	3,0	4,2
16642 POLICY-OTHER file URI scheme attempt	2	169,8	4,2	1,1	4,2	3,0	4,2
23408 OS-WINDOWS Microsoft Windows large imag..	2	169,8	5,6	1,1	5,6	3,0	5,6
24254 INDICATOR-COMPROMISE IP only webpage redirect attempt	2	169,8	5,5	1,1	5,5	3,0	5,5
	77	195,9	5,8	0,9	5,8	2,7	5,8
24378 POLICY-OTHER TCP packet with urgent flag attempt	31	3,2	55,3	0,0	32,6	0,0	37,1
	34	113,1	55,2	1,0	32,4	3,9	43,8
	77	195,9	57,7	0,9	32,1	2,7	38,8
24814 PROTOCOL-SNMP Samsung printer default community string	34	113,1	8,1	1,0	8,1	3,9	8,1
	164	97,9	18,5	1,7	4,0	3,4	9,5
29456 PROTOCOL-ICMP Unusual PING detected	31	3,2	56,0	0,0	53,2	0,0	53,9

Figura 7.11: Resultados clusterización K-means de evidencias de Dataset Normal vs Metasploit.

7.3 Clasificación.

La explicación de las técnicas de clasificación se introdujo en el capítulo 5, en la sección 5.3. En ella se comentaba los tipos de algoritmos implementados dentro de Spark MLlib, separando los algoritmos de clasificación multiclase de los algoritmos de clasificación binaria. En esta sección, los modelos *off-line* a calcular serán clasificadores binarios: ataque o no; a diferencia de los experimentos realizados sobre el *dataset* KDDCup99 que fueron hechos como clasificadores multiclase. En esta sección además se hará uso de las métricas para sistemas de clasificación introducidas en la sección 5.3.1.

7.3.1 Algoritmos Adicionales: SVM.

La mayor parte de los algoritmos fueron introducidos en el apartado de la sección 5.3, si bien, al tratar exclusivamente con clasificadores binarios se puede hacer uso de otros que no se utilizaron en el análisis multiclase del *dataset* KDDCup99.

El algoritmo de *Linear Support Vector Machines* o también conocido como SVM, es una técnica de clasificación y de regresión muy conocida y popular, que a diferencia de la Regresión Logística, no es un modelo probabilista, si no que se preciden las clases basándose en la evaluación del modelo como positivas o negativas. La función de transferencia es la de identidad, así pues los datos de salida predecidos siguen la ecuación 7.1.

$$y = w^T x \tag{7.1}$$

Entonces, si la evaluación de $w^T x$ es mayor o igual que el umbral de 0, el modelo del SVM asignará la evidencia a la clase 1, mientras que en caso contrario se asignará a la clase 0. El umbral es un parámetro del modelo y puede ser ajustado debidamente. La función de pérdida de los SVMs se conoce como *hinge loss* y se identifica en la ecuación 7.2.

$$max(0, 1 - y\ w^T\ x) \tag{7.2}$$

SVM es un algoritmo que trata de encontrar el vector de pesos apropiado que haga que las clases estén lo más separadas posible. Este algoritmo ha sido objetivo de multitud de estudios en la literatura y se concluye como un mecanimos que se desenvuelve adecuadamente en las tareas de clasificación. En la variante lineal que implementa Spark MLlib, adicionalmente permite abordar grandes volúmenes de datos.

7.3.2 **Eventos de Host.**

Para generar los modelos de clasificación del entorno de Host, se usan los mismos *datasets* que en las pruebas de clusterización, teniendo: normal, nessus y metasploit. Con todos ellos se construye un modelo completo sobre el que se va a realizar el aprendizaje y la cross-validación. En la preparación de los datos, como principales diferencias a los modelos de clusterización, se separan las IPs concretas del campo de localización, y se etiqueta como posible Ataque toda evidencia con nivel de alerta mayor o igual que 7. Así mismo, se calculan las características para los vectores de evidencias, obteniendo 82 "sidids" diferentes en todo el *dataset* y 8 localizaciones. Los datos en crudo se separan en 3 grupos tomando evidencias de forma aleatoria, uno para entrenamiento (80%), otro para cross-validación del modelo aprendido (10%) y finalmente el último para el testeo final (10%). Todo ello se puede ver en el código 7.3.

Código 7.3: Conteo de total de casos por categoría

```
...
 val rawNormal = sc.textFile(
 "hdfs://ec2-52-8-225-253.us-west-1.compute.amazonaws.com:9000/root/ossec/ossecNormal/CSV/*")
 val rawNessus = sc.textFile(
 "hdfs://ec2-52-8-225-253.us-west-1.compute.amazonaws.com:9000/root/ossec/ossecNessus/CSV/*")
 val rawMeta = sc.textFile(
 "hdfs://ec2-52-8-225-253.us-west-1.compute.amazonaws.com:9000/root/ossec/ossecMeta/CSV/*")

 val rawData = rawNormal.union(rawNessus).union(rawMeta)

 val data = rawData.map(buildDataSet(rawData))

 val Array(trainData, cvData, testData) = data.randomSplit(Array(0.8,0.1,0.1))
 trainData.cache()
 cvData.cache()
 testData.cache()

...
  def getIpLocation(location:String): String = {
        if(location.contains("->"))
              location.split("->")(0).split(" ")(1)
        else
        "192.168.1.133"
  }

  def getLevelAttack(level:String): Int = {
        if(level.toInt >= 7)
              1
        else
              0
  }
...
def buildDataSet(rawData: RDD[String]): (String => (String,LabeledPoint)) = {
       val splitData = rawData.map(_.split(','))
```

```
    val sidids = splitData.map(_(0)).distinct().collect().zipWithIndex.toMap // 82 sidids
    val locations =
         splitData.map(_(6)).map(getIpLocation).distinct().collect().zipWithIndex.toMap
         // 8 locations

    (line: String) => {
           val buffer = line.split(',').toBuffer
           val sidid = buffer(0)
           val label = getLevelAttack(buffer(1))
           val level = buffer(1)
           val location = getIpLocation(buffer(6))

           val newSididsFeatures = new Array[Double](sidids.size)
           newSididsFeatures(sidids(sidid)) = 1.0
           val newLocationsFeatures = new Array[Double](locations.size)
           newLocationsFeatures(locations(location)) = 1.0

           val vector = new scala.collection.mutable.ArrayBuffer[Double]

           vector.insertAll(0, newSididsFeatures)
           vector.insertAll(0, newLocationsFeatures)
           ("Attack:"+label + " Level:" + level, LabeledPoint(label,
                Vectors.dense(vector.toArray)))
    }
}
```

7.3.2.1 **Aprendizaje.**

Al igual que se hizo con el *dataset* de KDDCup, se tienen que encontrar los parámetros más adecuados para cada modelo. La mayor diferencia entre ambos experimentos es que las métricas de estimación de bondad del modelo son distintas, más sencillas a la hora de seleccionar un buen clasificador que las que se usan para modelos multiclase. En concreto se van a evaluar tres métricas: *Accuracy*: La precisión no es más que el conteo de casos positivos acertados, dividido el número total de casos; *Area under ROC*: El área debajo de la curva ROC igual a 1.0 sería el clasificador perfecto; *Area under PR*: Un área debajo de la curva *Precission - Recall* de 1.0 sería el clasificador perfecto.

Los resultados del aprendizaje realizado son los siguientes (marcado en amarillo el modelo elegido para la fase de validación):

- Algoritmo de *Decision Tree* en la tabla 7.4.
- Algoritmo de *Random Forest* en la tabla 7.5.
- Algoritmo de *Naive Bayes* en la tabla 7.6.
- Algoritmo de *SVM* en la tabla 7.7.

Cuadro 7.4: Resultados aprendizaje modelo para Host del algoritmo Decision Tree.

impurity	depth	bins	Train Accuracy	Train Area Under ROC	Train Area Under PR	CV Accuracy	CV Area Under ROC	CV Area Under PR
gini	10	100	0.9987	0.9877	0.9884	0.9990	0.9906	0.9911
gini	10	150	0.9987	0.9877	0.9884	0.9990	0.9906	0.9911
gini	10	300	0.9987	0.9877	0.9884	0.9990	0.9906	0.9911
gini	20	100	0.9999	0.9998	0.9998	0.9998	0.9988	0.9988
gini	20	150	0.9999	0.9998	0.9998	0.9998	0.9988	0.9988
gini	20	300	0.9999	0.9998	0.9998	0.9998	0.9988	0.9988
gini	30	100	1.0	1.0	1.0	1.0	1.0	1.0
gini	30	150	1.0	1.0	1.0	1.0	1.0	1.0
gini	30	300	1.0	1.0	1.0	1.0	1.0	1.0
entropy	10	100	0.9986	0.9873	0.9880	0.9990	0.9906	0.9911
entropy	10	150	0.9986	0.9873	0.9880	0.9990	0.9906	0.9911
entropy	10	300	0.9986	0.9873	0.9880	0.9990	0.9906	0.9911
entropy	20	100	0.9999	0.9997	0.9997	0.9998	0.9988	0.9988
entropy	20	150	0.9999	0.9997	0.9997	0.9998	0.9988	0.9988
entropy	20	300	0.9999	0.9997	0.9997	0.9998	0.9988	0.9988
entropy	30	100	1.0	1.0	1.0	1.0	1.0	1.0
entropy	30	150	1.0	1.0	1.0	1.0	1.0	1.0
entropy	30	300	1.0	1.0	1.0	1.0	1.0	1.0

Cuadro 7.5: Resultados aprendizaje modelo para Host del algoritmo Random Forest.

impurity	depth	bins	Train Accuracy	Train Area Under ROC	Train Area Under PR	CV Accuracy	CV Area Under ROC	CV Area Under PR
gini	10	100	0.9956	0.9582	0.9604	0.9950	0.9511	0.9536
gini	10	150	0.9965	0.9672	0.9689	0.9968	0.9686	0.9701
gini	10	300	0.9844	0.8574	0.8575	0.9857	0.8681	0.8629
gini	20	100	0.9997	0.9976	0.9977	1.0	1.0	1.0
gini	20	150	0.9997	0.9973	0.9974	0.9996	0.9965	0.9966
gini	20	300	0.9995	0.9977	0.9957	0.9995	0.9975	0.9954
gini	30	100	0.9999	0.9991	0.9992	1.0	1.0	1.0
gini	30	150	0.9996	0.9986	0.9968	0.9996	0.9998	0.9965
gini	30	300	0.9999	0.9992	0.9993	1.0	1.0	1.0
entropy	10	100	0.9871	0.8785	0.8831	0.9857	0.8615	0.8654
entropy	10	150	0.9968	0.9726	0.9714	0.9967	0.9707	0.9687
entropy	10	300	0.9971	0.9777	0.9736	0.9980	0.9824	0.9810
entropy	20	100	0.9998	0.9985	0.9986	0.9998	0.9988	0.9988
entropy	20	150	0.9996	0.9969	0.9970	0.9994	0.9941	0.9944
entropy	20	300	0.9997	0.9974	0.9976	0.9992	0.9930	0.9933
entropy	30	100	0.9996	0.9994	0.9963	0.9995	0.9997	0.9953
entropy	30	150	0.9898	0.9033	0.9084	0.9916	0.9174	0.9216
entropy	30	300	0.9998	0.9990	0.9990	0.9996	0.9965	0.9966

Cuadro 7.6: Resultados aprendizaje modelo para Host del algoritmo Naive Bayes.

modelType	lambda	Train Accuracy	Train Area Under ROC	Train Area Under PR	CV Accuracy	CV Area Under ROC	CV Area Under PR
multinomial	0.1	0.9999	0.9999	0.9995	0.9996	0.9998	0.9965
multinomial	0.5	0.9990	0.9995	0.9914	0.9984	0.9991	0.9853
multinomial	1.0	0.9982	0.9990	0.9834	0.9978	0.9988	0.9799
multinomial	2.0	0.9977	0.9988	0.9793	0.9977	0.9988	0.9788
multinomial	10.0	0.9932	0.9964	0.9430	0.9934	0.9965	0.9423
multinomial	20.0	0.9915	0.9946	0.9305	0.9903	0.9905	0.9182
multinomial	50.0	0.9891	0.9834	0.9099	0.9880	0.9837	0.8995
multinomial	100.0	0.9790	0.8876	0.8036	0.9796	0.9012	0.8067
multinomial	200.0	0.9784	0.8575	0.7903	0.9797	0.8671	0.7971
multinomial	500.0	0.9771	0.7925	0.7863	0.9780	0.7935	0.7855
multinomial	1000.0	0.9475	0.5	0.5262	0.9494	0.5	0.5252
bernoulli	0.1	0.9994	0.9997	0.9947	0.9989	0.9994	0.9897
bernoulli	0.5	0.9972	0.9985	0.9751	0.9971	0.9985	0.9735
bernoulli	1.0	0.9948	0.9973	0.9557	0.9949	0.9973	0.9545
bernoulli	2.0	0.9929	0.9963	0.9410	0.9931	0.9964	0.9405
bernoulli	10.0	0.9745	0.9865	0.8368	0.9728	0.9856	0.8252
bernoulli	20.0	0.9713	0.9849	0.8236	0.9696	0.9840	0.8125
bernoulli	50.0	0.9576	0.8872	0.6931	0.9576	0.8995	0.6986
bernoulli	100.0	0.9642	0.8798	0.7120	0.9634	0.8926	0.7128
bernoulli	200.0	0.9819	0.8575	0.8263	0.9838	0.8693	0.8397
bernoulli	500.0	0.9475	0.5	0.5262	0.9494	0.5	0.5252
bernoulli	1000.0	0.9475	0.5	0.5262	0.9494	0.5	0.5252

Cuadro 7.7: Resultados aprendizaje modelo para Host del algoritmo SVM.

numIterations	Train Accuracy	Train Area Under ROC	Train Area Under PR	CV Accuracy	CV Area Under ROC	CV Area Under PR
100	0.9831	0.8500	0.8435	0.9852	0.8656	0.8576
200	0.9831	0.8500	0.8435	0.9852	0.8656	0.8576
300	0.9831	0.8500	0.8435	0.9852	0.8656	0.8576
400	0.9831	0.8500	0.8435	0.9852	0.8656	0.8576
500	0.9831	0.8500	0.8435	0.9852	0.8656	0.8576
600	0.9831	0.8500	0.8435	0.9852	0.8656	0.8576
700	0.9831	0.8500	0.8435	0.9852	0.8656	0.8576
800	0.9831	0.8500	0.8435	0.9852	0.8656	0.8576
900	0.9831	0.8500	0.8435	0.9852	0.8656	0.8576
1000	0.9831	0.8500	0.8435	0.9852	0.8656	0.8576

7.3.2.2 **Validación.**

Una vez seleccionados los modelos de la fase de aprendizaje, se procede al aprendizaje de todos ellos con los grupos *training* y *crossvalidation*. Sobre este modelo se evaluará el grupo de *test* para cada uno de ellos, añadiendo a la tabla 7.8 específicamente los *datasets* de ataques para ver cómo se comporta cada modelo, así como los tiempos de cálculo inferencial en un clúster de 5 máquinas de Amazon EC2 del completo de evidencias del grupo.

Cuadro 7.8: Validación de modelos seleccionados para evidencias de Host.

		D. Tree	R. Forest	Naive	SVM
Aprendizaje	Tiempo	54.89s	74.43s	1.38s	59.84s
Test dataset	Accuracy	1.0	0.9998	0.9998	0.9834
	AUC (ROC)	1.0	0.9988	0.9999	0.8503
	Area PR	1.0	0.9988	0.9988	0.8380
	Tiempo	0.7111s	0.7398s	0.6926s	0.73s
Nessus dataset	Accuracy	1.0	1.0	1.0	0.9625
	AUC (ROC)	1.0	1.0	1.0	0.5364
	Area PR	1.0	1.0	1.0	0.3130
	Tiempo	0.4384s	0.4378s	0.4068s	0.39s
Metasploit dataset	Accuracy	1.0	0.9998	0.9997	0.9504
	AUC (ROC)	1.0	0.9993	0.9998	0.8144
	Area PR	1.0	0.9994	0.9988	0.8357
	Tiempo	0.3979s	0.3810s	0.3588s	0.34s

7.3.3 **Eventos de Red.**

Para generar los modelos de clasificación del entorno de Red, se siguen el mismo proceso que con los eventos de *host*, usando los mismos *datasets* que en las pruebas de clusterización: normal, nessus y metasploit. Con todos ellos se construye un modelo completo sobre el que se va a realizar el aprendizaje y la cross-validación.

En la preparación de los datos, como principales diferencias a los modelos de clusterización, se filtran las IPs locales (192.168.1.0/24), y se etiqueta como posible Ataque toda evidencia distinta de las tres alertas genéricas propiciadas por las reglas adicionales introducidas. Así mismo, se calculan las características para los vectores de evidencias, obteniendo:

- 5 protocolos diferentes de capa 4.
- 10 direcciones diferentes como origen de la red 192.168.1.0/24 (se

añade un campo adicional denomiado *other* cuando no pertenece al rango)

- 20 direcciones diferentes como destino de la red 192.168.1.0/24 (se añade el campo *other* igual que en el caso anterior)
- 13 flags diferentes de TCP.
- 5 tipos de TOS diferentes.
- 4 ICMP Code distintos.
- 6 ICMP Type.

Al igual que con los eventos de *host*, los datos en crudo se separan en 3 grupos tomando evidencias de forma aleatoria, uno para entrenamiento (80%), otro para validación del modelo aprendido (10%) y finalmente el último para validación (10%). En este caso no se incluye el código por extensión, pero es análogo al presentado en la clasificación de evidencias de *host*.

7.3.3.1 **Aprendizaje.**

Los resultados del aprendizaje realizado son los siguientes (marcado en amarillo el modelo elegido para la fase de validación):

- Algoritmo de *Decision Tree* en la tabla 7.9. Se señalan los dos mejores modelos que se han obtenido, con una buena precisión, pero buscando una correcta relación entre los datos de Cross-Validación para tener una cercanía a 1.0 en las curvas de area debajo de ROC y de PR.
- Algoritmo de *Random Forest* en la tabla 7.10. Así mismo, igual que para los árboles de decisión, se indican dos modelos adecuados para realizar la clasificación del tráfico de red en la tabla correspondiente.
- Algoritmo de *Naive Bayes* en la tabla 7.11. El resultado después de pasar la etapa de cross-validación es considerablemente bajo para este algoritmo con los datos de entrada. Es por ello que obtiene resultados poco mejores que un clasificador aleatorio, así que lo más aconsejable es no tenerle en cuenta.

- Algoritmo de *SVM* en la tabla 7.12. Exactamente igual que en el caso del algoritmo anterior, para los datos a clasificar, este modelo pese a tener una buena precisión, no sería muy fiable, así que sería poco aconsejable fiarse de sus veredictos.

Cuadro 7.9: Resultados aprendizaje modelo para Red del algoritmo Decision Tree.

impurity	depth	bins	Train Accuracy	Train Area Under ROC	Train Area Under PR	CV Accuracy	CV Area Under ROC	CV Area Under PR
gini	10	100	0.9892	0.8494	0.6198	0.9886	0.8282	0.5932
gini	10	150	0.9891	0.9147	0.6821	0.9893	0.8950	0.6692
gini	10	300	0.9892	0.8536	0.6245	0.9886	0.8337	0.5991
gini	20	100	0.9898	0.7729	0.5634	0.9861	0.6982	0.4145
gini	20	150	0.9898	0.7643	0.5563	0.9858	0.6815	0.3886
gini	20	300	0.9898	0.7638	0.5564	0.9858	0.6815	0.3892
gini	30	100	0.9900	0.7562	0.5531	0.9852	0.6618	0.3542
gini	30	150	0.9900	0.7555	0.5527	0.9853	0.6632	0.3565
gini	30	300	0.9900	0.7550	0.5522	0.9852	0.6618	0.3542
entropy	10	100	0.9892	0.8427	0.6143	0.9886	0.8074	0.5731
entropy	10	150	0.9892	0.8432	0.6149	0.9885	0.8074	0.5719
entropy	10	300	0.9892	0.8469	0.6184	0.9884	0.8073	0.5701
entropy	20	100	0.9898	0.7643	0.5560	0.9859	0.6774	0.3854
entropy	20	150	0.9898	0.7728	0.5638	0.9861	0.6969	0.4129
entropy	20	300	0.9898	0.7580	0.5513	0.9861	0.6775	0.3892
entropy	30	100	0.9900	0.7585	0.5551	0.9852	0.6659	0.3590
entropy	30	150	0.9900	0.7552	0.5523	0.9852	0.6632	0.3554
entropy	30	300	0.9900	0.7552	0.5525	0.9852	0.6632	0.3554

Cuadro 7.10: Resultados aprendizaje modelo para Red del algoritmo Random Forest.

impurity	depth	bins	Train Accuracy	Train Area Under ROC	Train Area Under PR	CV Accuracy	CV Area Under ROC	CV Area Under PR
gini	10	100	0.9890	0.8903	0.6565	0.9891	0.8727	0.6445
gini	10	150	0.9888	0.6443	0.4250	0.9882	0.6301	0.3928
gini	10	300	0.9888	0.6999	0.4725	0.9885	0.6925	0.4615
gini	20	100	0.9892	0.8344	0.6071	0.9879	0.7932	0.5481
gini	20	150	0.9893	0.8210	0.5951	0.9883	0.7837	0.5452
gini	20	300	0.9893	0.8332	0.6072	0.9881	0.7850	0.5434
gini	30	100	0.9894	0.7212	0.5071	0.9876	0.6879	0.4336
gini	30	150	0.9892	0.8341	0.6066	0.9886	0.8088	0.5744
gini	30	300	0.9893	0.7889	0.5658	0.9875	0.7446	0.4914
entropy	10	100	0.9888	0.6093	0.3989	0.9882	0.5955	0.3628
entropy	10	150	0.9888	0.7744	0.5417	0.9887	0.7618	0.5314
entropy	10	300	0.9891	0.8451	0.6143	0.9887	0.8199	0.5869
entropy	20	100	0.9893	0.7974	0.5729	0.9884	0.7423	0.5077
entropy	20	150	0.9892	0.7345	0.5139	0.9878	0.6866	0.4364
entropy	20	300	0.9893	0.8331	0.6073	0.9881	0.7905	0.5477
entropy	30	100	0.9894	0.8425	0.6176	0.9878	0.7821	0.5346
entropy	30	150	0.9894	0.8536	0.6279	0.9887	0.8227	0.5896
entropy	30	300	0.9892	0.8295	0.6013	0.9881	0.7906	0.5489

Cuadro 7.11: Resultados aprendizaje modelo para Red del algoritmo Naive Bayes.

modelType	lambda	Train Accuracy	Train Area Under ROC	Train Area Under PR	CV Accuracy	CV Area Under ROC	CV Area Under PR
multinomial	0.1	0.5781	0.6571	0.3807	0.5821	0.6736	0.3959
multinomial	0.5	0.5781	0.6571	0.3807	0.5821	0.6736	0.3959
multinomial	1.0	0.5781	0.6571	0.3807	0.5821	0.6736	0.3959
multinomial	2.0	0.5781	0.6571	0.3807	0.5821	0.6736	0.3959
multinomial	10.0	0.5781	0.6571	0.3807	0.5821	0.6736	0.3959
multinomial	20.0	0.5781	0.6571	0.3807	0.5821	0.6736	0.3959
multinomial	50.0	0.5781	0.6571	0.3807	0.5821	0.6736	0.3959
multinomial	100.0	0.5781	0.6571	0.3807	0.5821	0.6736	0.3959
multinomial	200.0	0.5781	0.6571	0.3807	0.5821	0.6736	0.3959
multinomial	500.0	0.5781	0.6572	0.3807	0.5821	0.6736	0.3959
multinomial	1000.0	0.5783	0.6572	0.3807	0.5824	0.6738	0.3959

Cuadro 7.12: Resultados aprendizaje modelo para Red del algoritmo SVM.

numIterations	Train Accuracy	Train Area Under ROC	Train Area Under PR	CV Accuracy	CV Area Under ROC	CV Area Under PR
10	0.9882	0.5	0.5058	0.9880	0.5	0.5059
25	0.9882	0.5	0.5058	0.9880	0.5	0.5059
50	0.9882	0.5	0.5058	0.9880	0.5	0.5059
100	0.9882	0.5	0.5058	0.9880	0.5	0.5059
200	0.9882	0.5	0.5058	0.9880	0.5	0.5059
400	0.9882	0.5	0.5058	0.9880	0.5	0.5059
800	0.9882	0.5	0.5058	0.9880	0.5	0.5059
1000	0.9882	0.5	0.5058	0.9880	0.5	0.5059

7.3.3.2 **Validación.**

Una vez seleccionados los modelos de la fase de aprendizaje, se procede al aprendizaje de todos ellos con los grupos *training* y *crossvalidation*. Sobre este modelo se evaluará el grupo de *test* para cada uno de ellos, añadiendo a la tabla 7.13 específicamente los *datasets* de ataques para ver cómo se comporta cada modelo, así como los tiempos de cálculo inferencial en un clúster de 5 máquinas de Amazon EC2 del grupo completo de evidencias.

Cuadro 7.13: Validación de modelos seleccionados para evidencias de Red.

		D. Tree g-10-150	**D. Tree e-10-150**	**R. Forest g-10-100**	**R. Forest e-30-150**	**Naive**	**SVM**
Aprendizaje	Tiempo	5.18s	5.10s	11.35s	28s	0.4417s	20.4052s
Test dataset	Accuracy	0.9870	0.9872	0.9874	0.9867	0.5699	0.9875
	AUC (ROC)	0.8068	0.8122	0.7289	0.7934	0.6631	0.5
	Area PR	0.5555	0.5639	0.4831	0.5383	0.3917	0.5062
	Tiempo	0.6660s	0.3586s	0.5069s	0.4495s	0.3134s	0.3020s
Nessus dataset	Accuracy	0.9949	0.9949	0.9946	0.9949	0.6146	0.9935
	AUC (ROC)	0.7534	0.7702	0.6743	0.7279	0.6521	0.5
	Area PR	0.5736	0.5874	0.5093	0.5603	0.3518	0.5032
	Tiempo	0.5385s	0.5277s	0.5459s	0.5711s	0.4867s	0.4646s
Metasploit dataset	Accuracy	0.9677	0.9678	0.9677	0.9681	0.6634	0.9666
	AUC (ROC)	0.8520	0.8441	0.7467	0.8517	0.7074	0.5
	Area PR	0.6242	0.6169	0.5210	0.6259	0.4169	0.5166
	Tiempo	0.5020s	0.5099s	0.5443s	0.5543s	0.4436s	0.4322s

7.4 Análisis en tiempo real.

Para el análisis en tiempo real de la información proporcionada por las fuentes de red y máquina (*host*) se va a hacer uso de la funcionalidad de Spark Streaming y se parte del axioma de que para que un ataque externo[1] sea realmente a notificar, debe estar presente tanto a nivel de host como a nivel de red. Los datos de las evidencias, tal y como se comentó en el capítulo 6 llegan a través de dos tópicos de Apache Kafka, que son leídos por sendos procesos de Spark Streaming y volcados a disco en formatos JSON y CSV. El modelo unificado que se propone en la presente tesis trabaja a este nivel, a nivel de *microbatch*, jugando con la ventana temporal deslizante que ofrece Spark Streaming. Un diagrama explicativo se puede encontrar en la figura 7.12.

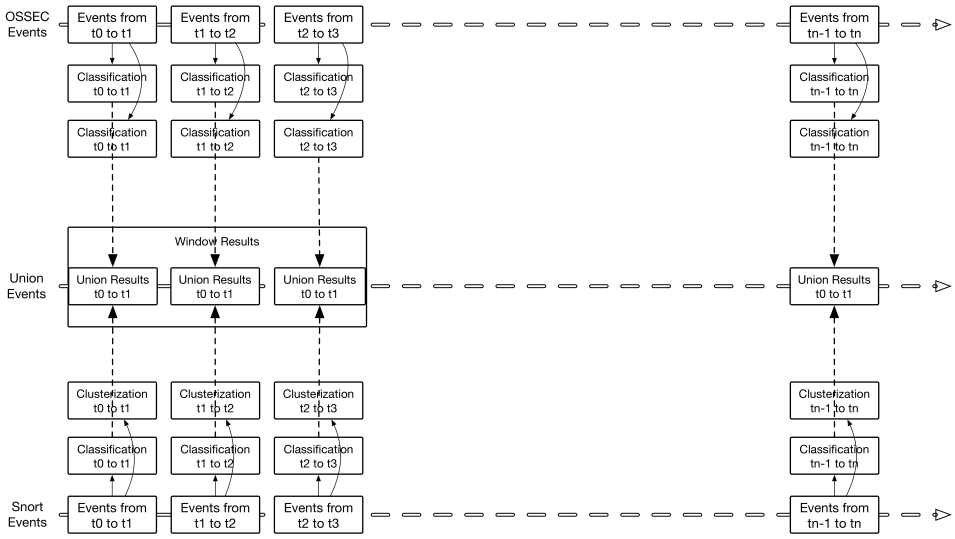


Figura 7.12: Explicación Streaming de Análisis en tiempo real.

A primera vista puede parecer un poco complejo el esquema, pero siguiendo un orden se va a tratar de explicar paso por paso, incluyendo fragmentos de código y resultados intermedios donde pudieran ser necesario.

1. En el capítulo 6 se recogen en flujos de micro-lotes de 1 segundo de duración los mensajes que se envían a los tópicos, tanto para Snort, como para Ossec. Estos mensajes están en formato JSON, así que el

[1] No proveniente del usuario local la máquina en cuestión.

DStream correspondiente se cambia mediante un map con una función de parseado a CSV. Estos son los últimos resultados realizados en el capítulo.

2. Partiendo de esa base, antes de poner en marcha la actividad de *Streaming*, el programa parte de que los modelos ya se encuentran cargados en el sistema de archivos HDFS, modelos *off-line* calculados en las secciones anteriores del presente capítulo. De estos modelos se debe hacer un *broadcast* para que estén presentes en todos los nodos del cluster.

Código 7.4: Carga de modelos calculados.

```
...
val ossecmodelkmeans = KMeansModel.load(sc,
     "hdfs://ubuntu1:9000/user/klobato/ossecmodelkmeans")
sc.broadcast(ossecmodelkmeans)
...
val snortmodelkmeans = KMeansModel.load(sc,
     "hdfs://ubuntu1:9000/user/klobato/snortmodelkmeans")
sc.broadcast(snortmodelkmeans)
...
val ossecmodelclassify = DecisionTreeModel.load(sc,
     "hdfs://ubuntu1:9000/user/klobato/ossecmodelclassify")
sc.broadcast(ossecmodelclassify)
...
val snortmodelclassify = DecisionTreeModel.load(sc,
     "hdfs://ubuntu1:9000/user/klobato/snortmodelclassify")
sc.broadcast(snortmodelclassify)
```

3. Para los modelos de K-means, dado que al final arrojan el cluster al que pertenece la evidencia y la distancia con el centroide del mismo, se realiza un paso adicional, para precalcular sobre la base de lo considerado "normal", la desviación máxima que se va a permitir en ese cluster. Para ello se calcula la media y la desviación estándar de los modelos K-means para Snort y para Ossec tomando como base el *dataset* normal. Lo primero es calcular la distancia de las evidencias normales frente a los centroides de cada cluster. Después ese RDD se usa para calcular $media + desviacintpica$, parámetro que será usado posteriormente para medir el grado de "sospechosa"que puede tener una evidencia.

Código 7.5: Precalculo de normalidad en clusteres sobre el dataset normal.

```
...
val clusterLabelDistSnort = normalDataSnort.map { case (loc, (label, datum)) =>
      val cluster = snortmodelkmeans.predict(datum)
```

```
        val dist = distToCentroid(datum, snortmodelkmeans)
        (cluster, label, dist)
        }
val clusterDistMediaSnort = clusterLabelDistSnort.map{case (c,l,d) =>
     (c,d)}.groupByKey().map{ case (c, dlst) =>
        (c, dlst.map(d => (d,1)))
        }.map { case (cluster, distCounter) =>
                            val s = distCounter.foldLeft((0.0, 0.0, 0)) {
                            (rTriple, caso) => (rTriple._1 + caso._1, rTriple._2 +
                                 (caso._1 * caso._1), rTriple._3 + caso._2)
                            }
                            (cluster, s._1, s._2, s._3)
                      }.map{ case (cluster, sum, sumSq, n) =>
                            val mean = sum/n
                            val stdev = if(n == 0) 0.0 else math.sqrt(n*sumSq -
                                 sum*sum) / n
                            (cluster, n, mean, stdev)
                      }.map{case (c,n,m,s) =>
                            (c, m+s)
                      }.collectAsMap()
sc.broadcast(clusterDistMediaSnort)
...
```

4. Una vez cargada toda la información necesaria, el siguiente paso es poner en marcha el sistema de Streaming, configurando dos flujos de Kafka a ambos tópicos y obteniendo un *DStream* a intervalos de 1 segundo, que según llegan crearán otro *DStream* con los datos transformados de JSON a CSV.

Código 7.6: Puesta en marcha del sistema de Streaming.

```
...
   val topicMapSnort = topicSnort.split(",").map((_,numThreads.toInt)).toMap
   val topicMapOssec = topicOssec.split(",").map((_,numThreads.toInt)).toMap
   val snortJsonLines = KafkaUtils.createStream(ssc, zkQuorum, groupSnort,
        topicMapSnort).map(_._2)
   val ossecJsonLines = KafkaUtils.createStream(ssc, zkQuorum, groupOssec,
        topicMapOssec).map(_._2)

   val snortCSVLines = snortJsonLines.map(parseSnortJson)
   val ossecCSVLines = ossecJsonLines.map(parseOssecJson)
...
```

5. El siguiente paso tanto para clasificadores como para clusterización, es adaptar la evidencia en formato CSV al formato adecuado para que el modelo pueda trabajar con ella (normalización, filtrado de parámetros, validación de los mismos, etc.). En concreto, en algunas categorías se ha introducido un campo especial denominado "unknown" por si llega alguna etiqueta no vista durante el aprendizaje. El resultado es otro *DStream* con tuplas de tuplas (se hace así por simplicidad), conte-

niendo la localización como clave de la primera tupla, la información del evento proveniente de la fuente, y los Vectores que requieren los motores de algoritmia:

Código 7.7: Creación del DStream de Evidencias desde CSV.

```
...
val ossecevidencekmeans: DStream[(String,(String,Vector))] =
     ossecCSVLines.map(buildOssecEvidenceKmeans(sidids,locations,_))
...
val ossecevidenceclassify: DStream[(String, (String, LabeledPoint))] =
     ossecCSVLines.map(buildOssecEvidenceClassify(sidids,locations,_))
...
val snortevidencekmeans: DStream[(String,(String,Vector))] =
     snortCSVLines.map(buildSnortDataSetKmeans(
l4_protos,srcs,dsts,tcpflagss,toss,icmpcodes,icmptypes,_))
...
val snortevidenceclassify: DStream[(String, (String, LabeledPoint))] =
     snortCSVLines.map(buildSnortDataSetClassify(
l4_protos,srcs,dsts,tcpflagss,toss,icmpcodes,icmptypes,_))
...
```

6. Hasta este momento se tienen 4 DStreams por separado, con origen común 2 a 2 del DStream que contiene los eventos para esta ventana temporal de 1 segundo.

7. Para los clasificadores, el siguiente paso es natural: solicitar a los motores propagados por los nodos que realicen su predicción. A modo de ejemplo, se muestra el código que permite ver los resultados en formato texto.

Código 7.8: Creación del DStream para Clasificadores Snort y Ossec.

```
    ...
val ossecevidenceclassify: DStream[(String, (String, LabeledPoint))] =
     ossecCSVLines.map(buildOssecEvidenceClassify(sidids,locations,_))
val resultsossecclasify = ossecevidenceclassify.map{ case (location:String,
     (label:String,datum:LabeledPoint)) =>
                    val veredict = ossecmodelclassify.predict(datum.features)
                    "Location:" +location+ " Label:\"" + label + "\" Classify:" +
                         veredict
}
val snortevidenceclassify: DStream[(String, (String, LabeledPoint))] =
     snortCSVLines.map(buildSnortDataSetClassify(l4_protos,srcs,dsts,tcpflagss,toss,icmpcodes,icmptypes,
val resultssnortclasify = snortevidenceclassify.map{ case (location:String,
     (label:String,datum:LabeledPoint)) =>
       val veredict = snortmodelclassify.predict(datum.features)
       "Location:" +location+ " Label:\"" + label + "\" Classify:" + veredict
}
    ...
```

8. El siguiente paso para los modelos de K-means, clusterización, aprendizaje no supervisado o detectores de anomalías, es realizar la adaptación de sus resultados a la regla que se ha establecido apriori: sospechoso todo aquello que se distancia del centroide más que la suma de la media y la desviación típica del *DataSet* normal.

Código 7.9: Creación del DStream para K-Means Snort y Ossec.

```
    ...
val resultsosseckmeans = ossecevidencekmeans.map{ case (location,(label, datum))
     =>
       val cluster = ossecmodelkmeans.predict(datum)
       val dist = distToCentroid(datum, ossecmodelkmeans)
       val suspicious = if(clusterDistMediaOssec(cluster) < dist) 1 else 0
       "Location:"+location+" Label:"+ label +" Cluster:"+ cluster +" Distance:" +
            dist + " Suspicious: " + suspicious
}
    val snortevidencekmeans: DStream[(String,(String,Vector))] =
         snortCSVLines.map(buildSnortDataSetKmeans(l4_protos,srcs,dsts,tcpflagss,toss,icmpcodes,icmptypes,_))
    val resultssnortkmeans = snortevidencekmeans.map{ case (location,(label,
         datum)) =>
           val cluster = snortmodelkmeans.predict(datum)
           val dist = distToCentroid(datum, snortmodelkmeans)
           val suspicious = if(clusterDistMediaSnort(cluster) < dist) 1 else 0
           "Location:"+location+" Label:"+ label +" Cluster:"+ cluster +"
                Distance:" + dist + " Suspicious: " + suspicious
    }
    ...
```

9. Finalmente, ya que hasta el momento se tienen 4 flujos por separado, el siguiente paso es unirlos para evaluar sus resultados. Pudiendo agrupar los eventos de los 4 motores y las 2 fuentes de información por localización.

10. Es más, haciendo uso de las funcionalidades de Spark Streaming, se pueden anexar en una ventana temporal de mayor tamaño, haciendo que sólo se lancen alertas bajo las premisas deseadas.

Para verificar el experimento se ha lanzado la herramienta de Metasploit [Rap15], figura 7.13 sobre el laboratorio de trabajo de nuevo, generando tanto una ventana de evidencias de 30 segundos a imprimir cada 10, como el formato unificado completo de todas las fuentes de información. Los resultados de detección son los esperados por cada uno de los motores introducidos en el modelo y han sido presentados en las secciones anteriores. Como nota, el motor de detección de anomalías funciona estupendamente en el caso de OSSEC, pero no tan bien en el caso de Snort, posiblemente debido al aprendizaje condicionado que se ha realizado en este ámbito,

mucho más abierto a la recogida de evidencias.

Lo que se realiza con el código de ejemplo anterior en este experimento, es el hecho de aglutinar en una ventana temporal de la duración deseada, todos los eventos recibidos, bien por Snort, bien por Ossec, con la calificación correspondiente que asignen los modelos de machine learning. La premisa para reducir el número de alarmas se basa en que sólo se notifican aquellas en las que se producen tanto en Red como en Host, reduciendo así el número de falsos positivos.

Como conclusiones finales, se puede indicar que sí es cierto el axioma de que se reduce el número de alertas cuando se busca que haya correlación entre Host y Red a la hora de generar el aviso o la correspondiente respuesta activa, si bien siempre queda el peligro de que la fuente de recogida de información de host no detecte el ataque.

Para concluir esta fase de experimentación, se debe indicar la fortaleza del modelo planteado y las diversas posibilidades que ofrece, pudiendo configurarse tanto la ventana temporal, como las fuentes o reglas en las que lanzar una alerta o una respuesta activa en mera combinatoria. Ejemplos de ello: cuando en la ventana temporal, para un host concreto, sólo se alerta si ambos clasificadores han identificado una alerta, o en el caso de que uno de ellos haya identificado más de una y el modelo de detección de anomalías considere alguna evidencia como sospechosa, etc.

En ultima instancia se debe señalar, que los tiempos en los que se están ejecutando dos clasificadores para Snort (Decision Tree y Random Forest), dos K-means (uno para Snort y otro para Ossec) y 4 clasificadores de OSSEC, son realmente buenos, como se puede apreciar en la figura 7.13 cumpliendo con los objetivos de tiempo real deseados en la presente tesis para el modelo propuesto. En la gráfica se observan dos picos, uno es el acumulado antes de lanzar las herramientas de envío de mensajes a Kafka, que realizan el primer envío a la máxima velocidad para llegar hasta la posición en la que se encuentran escribiendo Snort y Ossec, y el otro es cuando se lanza la prueba de escaneo profundo de Metasploit, pero como se aprecia tiende a reducir los tiempos de planificación y a la estabilidad general del modelo.

Figura 7.13: Apache Spark Streaming con todos los modelos bajo ataque de Metasploit.

«Si no sabes cómo hacer la pregunta correcta, no descubrirás nada.»
W. Edward Deming(1900 - 1993)

Capítulo
8
Conclusiones

Durante esta tesis se han introducido modelos, tecnologías, desarrollos y experimentos para la resolución de diversos problemas del área de Sistemas de Detección/Prevención de Intrusiones dentro del campo de la Seguridad de la Información. El presente capítulo resume los resultados de esta investigación y trata de cuantificar los objetivos alcanzados a lo largo de la misma. El resto del capítulo se organiza como sigue: la sección 8.1, enumera la consecución de los principales objetivos logrados en esta tesis; la sección 8.2, discute las limitaciones de la investigación realizada; la sección 8.3, describe las siguientes líneas de trabajo que pueden darse a través de los hitos alcanzados en esta tesis; finalmente, la sección 8.4, señala las observaciones finales sobre los puntos más representativos del presente escrito.

8.1 Principales contribuciones

En los capítulos previos se han presentado diferentes alternativas para resolver varios problemas relativos al área de Sistemas de Detección de Intrusiones, dentro del campo de la Seguridad de la Información. Volviendo a retomar la hipótesis de partida establecida en el primer capítulo:

> «*Existe un modelo, que trabajando en tiempo real, permite correlacionar evidencias producidas dentro del sistema opera-*

> *tivo y de la red simultáneamente para detectar ataques a la Seguridad la Información. Este modelo, integraría tanto la Detección de Usos Indebidos como la Detección de Anomalías, incluyendo para ambos casos el Conocimiento Experto dentro de una capa de Inteligencia Artificial.*»

Resumiendo las contribuciones principales presentadas en esta tesis desde el capítulo 3 hasta el capítulo 7:

1. **Un modelo unificado que puede integrar las soluciones actuales comerciales y de investigación en el Área de Detección de Intrusiones.**

   Se ha propuesto un modelo basado en Big Data, que es completamente integrable con las soluciones actuales (capítulo 4), complementando su actividad e incluso pudiendo llegar a tomar autonomía propia como motor de análisis. El modelo es unificado ya que sigue la filosofía de los sistemas SIEMs, en los que el número de fuentes es configurable, llevando las mismas técnicas de análisis a todas ellas. En concreto a lo largo de esta tesis se usan dos fuentes heterogéneas, evidencias de máquina (Host) o de red (Network), y se unifican las evidencias proporcionadas por ambas de forma satisfactoria.

2. **El modelo planteado parte de la base del procesamiento de grandes volúmenes de información en Tiempo Real, basándose en soluciones Big Data.**

   Acometer la recuperación, almacenamiento y procesamiento de grandes volúmenes de información es una premisa principal en la tesis. Tanto es así que se dedica un capítulo experimental completo (capítulo 6) a la capa de recogida de información en tiempo real. Se han hecho desarrollos y modificaciones específicas para que los sensores principales del SIEM *opensource* OSSIM, entreguen la información a una capa de recepción totalmente escalable basada en Apache Kafka. Así mismo, todos los experimentos de *machine-learning* efectuados han sido medidos para dar respuesta a las evidencias siguiendo la filosofía del modelo en tiempo real de la presente tesis.

3. **El modelo procesa fuentes heterogéneas, incluyendo el conocimiento experto de la comunidad de Seguridad, sin penalizar la detección de ataques nóveles.**

La base de los experimentos y del modelo propuesto es tomar el conocimiento expresado en reglas de los sistemas de recolección de evidencias. Las evidencias llegan en crudo, si bien caso de que activen alguna regla, ésta se traspasa con la evidencia completa. Es fundamental para entrenar una capa de algoritmos de *machine-learning* que tomen el conocimiento que los expertos han arrojado a las herramientas de detección, tanto de host como de red. Ambas fuentes heterogéneas son recibidas por la capa de adquisición y trasladadas convenientemente a los modelos de machine-learning planteados. Así mismo, se crean para el entorno sendos detectores de anomalías, tanto para el tráfico de red (network), como para las evidencias provenientes de host (máquina), que permiten el reconocimiento de toda actividad subversiva fuera de lo considerado como normal para el entorno.

4. **Se propone una composición de un laboratorio orientado a experimentación, replicable y reutilizable para las pruebas y validación tanto de la escalabilidad en número de fuentes usadas, como en el tipo de ataques que permite realizar.**

   Una de las mayores dificultades a la hora de realizar este tipo de investigaciones es la generación de un *dataset* apropiado. Ya se discutió en el capítulo 3 que existen varios *datasets* que pueden usarse y se utilizan comúnmente en investigación, si bien no es la mejor forma de realizarla, ya que en general parten de datos y ataques lejos de los actuales. Es por ello que se prepara un laboratorio propio, que permita tanto la explotación de vulnerabilidades, como la reproducibilidad en un entorno controlado de las evidencias. Desde el punto de vista de la capa de Big Data, este laboratorio propuesto, complementado con máquinas de Amazon EC2, permite una revisión de la situación muy detallada, pudiendo organizar la investigación debidamente en cada una de sus etapas.

5. **Se propone una algoritmia combinada basada en clusterización (aprendizaje no supervisado) y clasificación (aprendizaje supervisado), Metaclasificación.**

   La capa de *machine-learning* empleada usando el modelo planteado, es completamente escalable variando exclusivamente el número de máquinas y pudiendo afrontar tanto flujos de datos en tiempo real, como históricos de los eventos acontecidos durante años en formato crudo. La combinación de las evidencias de las fuentes heterogéneas

se realiza temporalmente, haciendo hincapié en los axiomas genéricos planteados en origen: la correlación entre evidencias de red y evidencias de host. El capítulo 7 se dedica en exclusiva a la creación de los modelos off-line y on-line de clasificación (aprendizaje supervisado) y clusterización (aprendizaje no-supervisado), proponiendo una nueva metodología de Metaclasificación temporal.

6. **Se plantea una solución capaz de afrontar ataques nóveles (Zero Days).**

   Como resultado de unificar fuentes heterogéneas, así como modelos de aprendizaje supervisado y no-supervisado, está el hecho de que por un lado se detectarán todas aquellas evidencias que sean un ataque efectivo (combinación de evidencias de host con evidencias de red) basado en conocimiento experto reflejado en el aprendizaje, y todas aquellas evidencias combinadas que se salgan de los modelos de aprendizaje no supervisado, siendo este punto crítico para la detección de ataques nóveles. Los ataques nóveles serán detectados bien por semejanza con ataques anteriormente aprendidos, o bien por salirse de la normalidad aprendida (modelos de aprendizaje no supervisados). En el capítulo 7 se detallan las bondades y los resultados experimentales de esta nueva aproximación.

7. **El modelo planteado busca que la escalabilidad lineal en base al número de máquinas del clúster**

   Todos los desarrollos de nuevo modelo de Big Data presentados han tenido en cuenta la escalabilidad de cada una de sus partes. Se ha validado durante todos los experimentos que se mantenga la premisa de que toda carga de trabajo, ya sea de recepción o de procesamiento, se mantenga escalable, y que esta escalabilidad sea lineal en relación al número de máquinas empleadas en el cluster de Big Data.

Con la consecución de las contribuciones anteriores se ha conseguido cumplir con los objetivos específicos establecidos en la sección 1.3.3. Cuyo objetivo principal definido es:

> «*Creación de un nuevo modelo Big Data para el área de la Seguridad de la Información, que se integre con las soluciones actuales, y que permita aplicar técnicas de Inteligencia Artificial en Cascada correlacionando evidencias de fuentes heterogéneas (Host, Red, etc.), con respuesta en tiempo quasi*

real y que tenga escalabilidad en base al número de máquinas usadas en el cluster»

Definiendo los siguientes objetivos específicos:

1. *Definición de la arquitectura más conveniente para abordar el objetivo principal.*

   Se define en la sección 4.11 del capítulo 4, dando incluso varios niveles de implantación e integración con los Sistemas SIEMs actuales.

2. *Selección de las tecnologías de Big Data que mejor puedan alojar al modelo planteado en el objetivo principal.*

   Para la capa de recepción, a parte de desarrollos propios, se utiliza Apache Kafka [Fou15h]. Como motor de análisis Big Data se hace uso de Apache Spark [Fou15a], usando el lenguaje Scala [Sca15], así como de HDFS [Fou15m] como sistema de persistencia.

3. *Búsqueda y generación de evidencias adecuadas para validar el modelo en cuestión, teniendo en cuenta tráfico normal y ataques dirigidos.*

   Se genera un laboratorio propio donde realizar todas las pruebas de ataques y recolección de evidencias, usando las dos fuentes principales de información del sistema SIEM OSSIM: Snort y Ossec, explicado en los capítulos 3 y 6.

4. *Selección, despliegue y desarrollo de los sistemas que permitan la recolección de evidencias de host.* (Capítulos 3 y 6)

5. *Selección, despliegue y desarrollo de los sistemas que permitan la recolección de evidencias de red.* (Capítulos 3 y 6)

6. *Establecimiento de una metodología para aplicación de las tecnologías Big Data al procesamiento de la información anterior, permitiendo identificar las variables más representativas de la capa de adquisición.* (Capítulos 6 y 7)

7. *Localizar una combinación de algoritmos con una tasa de falsos positivos/negativos baja aceptable que pueda ser aplicado en el campo de la Seguridad de la Información.* (Capítulo 7)

Quedando cumplidos los objetivos operacionales establecidos:

1. Diseñar e implementar un mecanismo de captura de evidencias de bajo nivel del sistema operativo.

2. Diseñar e implementar un mecanismo de captura de evidencias de bajo nivel en el entorno de red.

3. Creación de un modelo de datos apropiado para poder recoger la información de los mecanismos anteriores: la capa de adquisición.

4. Diseñar e implementar un sistema capaz de recorrer la información recabada en tiempo real, permitiendo el acceso a los datos recogidos.

5. Estudio teórico de los diferentes algoritmos que se puede aplicar sobre el modelo de datos anterior buscando el tiempo óptimo de procesamiento por cada nivel del clúster.

6. Diseño e implementación de los algoritmos seleccionados para crear una clusterización jerárquica por niveles capaz de acometer la detección de anomalías.

Con la satisfacción de los objetivos específicos y operacionales, se considera que se ha cumplido el objetivo principal de la presente tesis, así como el hecho de que el trabajo realizado valida la hipótesis inicial establecida.

8.2 Discursión de las principales contribuciones

En cada capítulo se describe tanto las contribuciones que pueden aportarse, como los objetivos parciales y totales que se buscan, así como las limitaciones que existen en cada contribución a esta tesis. No obstante en esta sección se van a resumir las mayores limitaciones a las principales contribuciones.

La primera parte de la presente tesis trata de establecer la captura de información desde fuentes heterogéneas y la arquitectura más apropiada para conducir el desarrollo del nuevo modelo. En general, ambos temas adolecen del mismo problema: la generación de vectores de ataques y de tráfico normal. Si bien en la literatura se siguen usando los *datasets* públicos, como el KDDCup99, usado en el capítulo 5, no son lo bastante actuales como para poder realizar afirmaciones categóricas sobre la bondad o no de los ataques evidenciados. Así mismo, el no disponer de una infraestructura

grande sobre la que desplegar la solución, si bien no se estima problemático, dada la escalabilidad de base que se ha buscado desde el principio, sí que podría aportar valor a la generación de modelos de clasificación y clusterización más ricos en evidencias. Sería muy interesante poder desplegar la solución de la presente tesis en un entorno corporativo grande, con muchos más equipos y redes a las que monitorizar, pudiendo validar completamente el escalado correcto del modelo planteado, así como generar modelos sobre más volumen de datos de todas las posibles clases.

Para solventar la limitación anterior, se ha trabajado en un laboratorio donde realizar las actividades de *pentesting*, usando herramientas tradicionales en la auditoría de los sistemas de información. En este laboratorio se han desplegado versiones de sistemas operativos vulnerables para poder permitir a los ataques tener el impacto deseado, si bien, no se trata de ningún ataque de tipo Zero-Day. Los modelos de normalidad aprendidos se basan en el tráfico de un entorno controlado, por lo que los resultados son muy buenos en la fase experimental (técnicas K-means de clusterización). Aun siguiendo el mismo principio de aprender lo normal, en entornos de mayor número de equipos, servidores y segmentos de red, se tendría una base más sólida para evaluar la bondad de los modelos aplicados.

Siguiendo la línea de pensamiento de la comunidad *hacker*, sabiendo que existen modelos de normalidad, con anterioridad ya se han establecido mecanismos para burlar a los mismos, haciendo que el aprendizaje sea conducido poco a poco. En este punto, la presente tesis no trabaja, haciendo que el corte de lo que es normal o no sea establecido por conocimiento experto. Exactamente igual, para los modelos de aprendizaje supervisado, se han realizado experimentos en un entorno controlado, enseñando al sistema de forma guiada por conocimiento experto del autor en la materia. Un área difícil de afrontar es el establecer la bondad de los modelos aprendidos de forma automática, haciendo que sea imposible o dificultando las técnicas mencionadas, conocidas como *eavesdropping*.

Los modelos establecidos de machine-learning tanto para evidencias hosts, como para evidencias de red, no hacen uso de toda la información que reciben, requiriéndose otro tipo de técnicas para el procesamiento y generación de modelos más apropiados. Entre ellas estarían los sistemas de análisis de texto, pudiendo establecer en lenguaje formal la comunicación de los protocolos de red, obviamente quitando la capa de cifrado correspondiente. Siguiendo la línea anterior, de la capa de host, sólo se reciben alertas de las reglas ya establecidas, con el modelo planteado sería viable realizar estudios de los logs al completo, obteniendo mayor información

sobre los hechos acaecidos.

Entrando más a fondo en la recogida de evidencias, siguen existiendo mecanismos para hacer que las herramientas seleccionadas dejen de monitorizar y más si se ha conseguido comprometer la seguridad física de los equipos en cuestión. Esto haría que las fuentes como OSSEC, comprometidas con *rootkits* dejasen de escribir los logs que se usan para enviar la información, o incluso parasen los servicios de entrega de conocimiento a la nube. De igual forma, los ataques más comunes que hacen vulnerable a la fuente de tráfico de red, Snort, suelen venir derivados de la generación de muchas evidencias haciendo que el sistema de recolección no pueda procesar tantos paquetes y deje de actuar en la forma esperada.

En la fase de recogida de evidencias de hosts, inicialmente se planteaba la creación de un driver de bajo nivel que pasase información de ficheros, conexiones de red, procesos en ejecución y dependencias del ejecutable con registro u otros archivos. Esta línea de trabajo no descrita en el presente documento, tuvo que abandonarse dada la ingente cantidad de datos que se generaban, incluso aun aplicando ciertos filtros, era inviable poder abordar el procesamiento de esta información en crudo, llegando a saturar el ancho de banda de la tarjeta de red para exportar la información hacia afuera.

Los experimentos realizados en la segunda fase, si bien son representativos y adecuados para la investigación realizada, pueden haberse visto muy condicionados al entorno de trabajo preparado, ofreciendo resultados diferentes respecto a la bondad de los algoritmos en entorno real, requiriendo ajustes y/o trabajo adicional para hacerlos productos explotables.

8.3 Futuras líneas de investigación

En los siguientes puntos se identifican las posibles líneas de investigación futuras que puedan solventar los retos y problemas establecidos en el punto anterior, o bien, incluso ir un paso más allá continuando la investigación realizada en la presente tesis.

1. **Big Data Based Protocol Intrusion Detection System (PIDS)** Uno de los puntos no abordados en la presente tesis, pero que puede ser tratado a futuro, es el uso de los datos de los paquetes recogidos para poder establecer el protocolo, detectando conversaciones normales o anómalas en los propios datos. Para ello, por ejemplo, se puede hacer uso de las técnicas TF-IDF de procesamiento del lenguaje natural, estableciendo la gramática más adecuada para cada tipo de protocolo y

servicio, ajustándose al tráfico que se intercambia.

2. **Data Fusion** Las técnicas de *Data Fusion* se ocupan de aunar o combinar información de fuentes heterogéneas, de forma que sobre el resultado se pueda operar para conseguir un valor añadido que no se tenía con las fuentes por individual. Este campo tiene numerosas técnicas y deberá ser el más explotado en el área de la Seguridad de la Información, sobre todo con la cantidad de información que podría llegar a recogerse sobre modelos basados en el paradigma señalado en la presente tesis (Big Data).

3. **Reduce False Positive Rate. Correlation Techniques**. Estas técnicas se aplicarían a los eventos recibidos desde fuentes heterogéneas y se basan en la aplicación de la Teoría de la Probabilidad entre los eventos acaecidos, tratando de encontrar relaciones de dependencia entre ellos. Estas técnicas que ya se usan a día de hoy, siguen constituyendo una fuente de investigación a nivel de eventos de seguridad, tratando de mitigar los ratios de Falsos Positivos que ocurren actualmente en los sistemas de detección de intrusiones.

4. **Reduce False Positive Rate. Recommendation Systems.** Una técnica que se está explorando es el concepto de comunidad de expertos, es decir, dar una serie de recomendaciones en función de situaciones ya acaecidas a las que se les ha dado solución. Los sistemas de recomendación, fácilmente paralelizables, permiten observar las acciones tomadas por los expertos a lo largo del tiempo para una cadena de evidencias de entrada y ofrecer una salida en función de la resolución que se haya dado a casos semejantes. Es un campo prometedor, pero con dependencia de que las decisiones, respuestas, se hayan tomado de forma adecuada.

5. **Real Time Requeriments** Para los sistemas basados en dectección de usos indebidos, el proceso de encontrar la firma sobre las evidencias es clave a la hora de limitar su rendimiento, siendo en modelos de Big Data un cuello de botella importante a hacer frente. Para los sistemas basados en detección de anomalías es más complicado el análisis del gran volumen de información ya que depende de la complejidad del modelo utilizado, pero se deberá trabajar en modelos matemáticos paralelizables como los mostrados en la presente tesis.

6. **New Business Intelligence Tools for Security Dashboards.** En este escrito se ha usado ampliamente Spark SQL, Hive y el software de Tableau para análisis de la información. De igual forma existen herramientas como ElasticSearch o SOLR que permiten realizar esta actividad de indexación y cuantificación tipo *Business Intelligence*, si bien hasta el momento no hay ninguna que se ocupe exclusivamente de los eventos de seguridad, generados en forma de ráfagas muchas veces, y que deben asesorar en la generación de cuadros de mando para la toma de decisiones a los expertos en el campo.

7. **Big Data Security.** Uno de los campos a futuro para los Sistemas de Detección de Intrusiones que deberán llevarse a cabo es la protección de los mismos sistemas. El paradigma de computación cambia con modelos como el de la presente tesis y conllevan un esfuerzo adicional en la protección tanto de los clústeres de máquinas como de la información que mantienen. En este aspecto, como línea de investigación, la seguridad del entorno Big Data, seguro que jugará un papel fundamental para garantizar la privacidad de acceso a los datos y la seguridad de las infraestructuras.

8.4 Notas Finales

El salto a los modelos de Big Data es una necesidad para los Sistemas de Detección de Intrusiones, si bien hasta la fecha, pocos fabricantes han dado el paso, con excepción de Splunk [Spl15].

El entorno de trabajo creado por Apache Foundation para tratar con Big Data realmente es espectacular, facilitando la adopción de nuevas áreas de trabajo, como las tratadas en la presente tesis. El uso de lenguajes funcionales como Scala [Sca15], ha simplificado enormemente la programación paralela de las capas de recepción o de la generación de los motores de machine learning. El ecosistema de Hadoop, junto con Spark, constituyen un mecanismo muy potente de poder tratar información heterogénea de forma sencilla y ágil.

Así mismo, OSSIM [Ali15], de Alienvault, realiza un trabajo espléndido a la hora de recoger información de seguridad, permitiendo a herramientas *open-source* como Snort [aiA15] u OSSEC [Mic15] dar todo su potencial. La transferencia de información a tiempo real a Apache Kafka, sí que ha resultado complicado en algunos momentos para poder establecer flujos adecuados de información, aunque es muy potente y totalmente escalable

en el número de fuentes, se tiene que configurar adecuadamente para dar un mantenimiento a futuro.

Como herramientas de *pentesting* se han usado Metasploit [Rap15] y Nessus [tns15]. Ambas han sufrido una profesionalización bastante significativa en estos últimos años a la hora de poder servir de ayuda en la auditoría de sistemas, haciéndose relativamente accesibles al público en general. Si bien, se debe mencionar que las versiones útiles son las de pago, dejando las versiones libres sólo para usos puntuales.

Los resultados de la presente tesis han sido satisfactorios, demostrando tanto la hipótesis como los objetivos, si bien, el área de estudio es muy cambiante, y conforme evolucionen los sistemas de Detección/Prevención de Intrusiones hacia el área de Big Data, el problema se cambiará de emplazamiento, teniendo que securizar este entorno principalmente, antes de poder garantizar una protección real al público en general.

Bibliografía

[20113] "NIST-SP 500-291, NIST Cloud Computing Standards Roadmap". CreateSpace Independent Publishing Platform (2013).

[AA12] Idoia Aguirre and Sergio Alonso. Improving the automation of security information management: A collaborative approach. *Security & Privacy, IEEE* **10**(1), 55–59 (2012).

[AAMP03] K. Anagnostakis, S. Antonatos, E. Markatos, and M. Polychronakis. E2xb: A domain-specific string matching algorithm for intrusion detection. *Proceeding of the 18th IFIP International Information Security Conference* (Mayo 2003).

[ACFW00] J. Allen, A. Christie, and Pickel J. y Stoner E. Fithen W., McHugh J. State of the practice of intrusion detection technologies. *Carnegie Mellon University Software Engineering Institute, Technical Report* (2000).

[ACN03] Pedro Alipio, Paulo Carvalho, and José Neves. Using clips to detect network intrusions. In "Progress in Artificial Intelligence", pages 341–354. Springer (2003).

[AFM00] W. A. Arbaugh, W. L. Fithen, and J. McHugh. Windows of vulnerability: A case study analysis. *IEEE Computer* (December 2000).

[AFTV94] D. Anderson, T. Frivold, A. Tamaru, and A. Valdés. Next generation intrusion detection expert system (nides), software users manual. *Computer Science Laboratory, SRI International* (Mayo 1994).

[AG99] M. Schatz A.K. Ghosh, A. Schwartzbard. Using program behavior profiles for intrusion detection. *3rd SANS Workshop on Intrusion Detection and Response* (1999).

[AGMG03] N. Abouzakhar, A. Gani, and Abuitbel M. y King D Manson G. Bayesian learning networks approach to cybercrime detection. *Post Graduate Symposium, John Moore University, Liverpool, UK* (Junio 2003).

[AHLR07] Chris Anley, John Heasman, Felix Linder, and Gerardo Richarte. "The Shellcoder's Handbook: Discovering and Exploiting Security Holes". Wiley John + Sons, Indianapolis, IN, edición: 2nd ed. edition (2007).

[aiA15] Cisco and/or its Affiliates. Snort (2015).

[AJA15] Omar Al-Jarrah and Ahmad Arafat. Network intrusion detection system using neural network classification of attack behavior. *Journal of Advances in Information Technology Vol* **6**(1) (2015).

[AKBD15] H Akramifard, L Mohammad Khanli, MA Balafar, and R Davtalab. Intrusion detection in the cloud environment using multi-level fuzzy neural networks. In "Proceedings of the International Conference on Security and Management (SAM)", page 75. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp) (2015).

[Ali15] AlienVault. Alienvault ossim: The world's most widely used open source siem (2015).

[ALL13] CLOUD SECURITY ALLIANCE. Big data analytics for security intelligence. Technical Report, Cloud Security Alliance (2013).

[Ama15a] Amazon. Amazon ec2 (2015).

[Ama15b] Amazon. Amazon ec2 - amazon aws management (2015).

[And80] J. Anderson. Computer security threat monitoring and surveillance. *Technical Report, Fort Washington, Pennsylvania* (Abril 1980).

[Axe00] S. Axelsson. Intrusion detection systems: A survey and taxonomy. *Technical report 99-15, Department of Computer Engineering, Chalmers University. Goteborg, Suecia* (2000).

[BAC+02] S. Beattie, S. Arnold, C. Cowan, P. Wagle, C. Wright, and A. Shostack. Timing the application of security patches for optimal uptime. *In Proceedings of the 2002. USENIX Systems Administration Conference (LISA).* (November 2002).

[BAMF01] H. K. Browne, W. A. Arbaugh, J. McHugh, and W. L. Fithen. A trend analysis of exploitations. *In Proceedings of the 2001 IEEE Symposium on Security and Privacy* (May 2001).

[Bas99] Tim Bass. Multisensor data fusion for next generation distributed intrusion detection systems. (1999).

[Bas00] Tim Bass. Intrusion detection systems and multisensor data fusion. *Communications of the ACM* **43**(4), 99–105 (2000).

[BCA10] Rainer Bye, Seyit Ahmet Camtepe, and Sahin Albayrak. Collaborative intrusion detection framework: Characteristics, adversarial opportunities and countermeasures. In "CollSec" (2010).

[BCH+01] E. Bloedorn, A. Christiansen, W. Hill, C. Skorupka, L. Talbot, and J. Tivel. Data mining for network intrusion detection: How to get started. *The MITRE Corporation, Technical Report* (2001).

[Beg14] Rachid Beghdad. K-means for modelling and detecting anomalous profiles. *International Journal of Computing* **6**(1), 59–66 (2014).

[BG12] Hetal Bhavsar and Amit Ganatra. Variations of support vector machine classification technique: a survey. *International Journal of Advanced Computer Research* **2**(4) (2012).

[BGM02] Massimo Bernaschi, Emanuele Gabrielli, and Luigi V Mancini. Remus: a security-enhanced operating system. *ACM Transactions on Information and System Security (TISSEC)* **5**(1), 36–61 (2002).

[BH00] B. R. Buck and J. Hollingsworth. An api for runtime code patching. *Journal of High Performance Computing Applications, Vol. 14(4), pp. 317-329.* (2000).

[BLB+12] Rafae Bhatti, Ryan LaSalle, Rob Bird, Tim Grance, and Elisa Bertino. Emerging trends around big data analytics and security: panel. In "Proceedings of the 17th ACM symposium on Access Control Models and Technologies", pages 67–68. ACM (2012).

[BM01] R. Bace and P. Mell. Intrusion detection systems. *NIST Special Publication on Intrusion Detection Systems. Disponible en http://csrc.nist.gov/publications/nistpubs/* (2001).

[BM02] Gabrielli E. Bernaschi, M. and L. V. Mancini. Remus: a security-enhanced operating system. *ACM Transactions on Information and System Security 5, 36 (Feb.)* (2002).

[BP04] Z. Baker and V. Prasanna. Automatic synthesis of efficient intrusion detection systems on fpgas. *Proceedings of the International Conference on Field-Programmable Logic and its Applications, Bélgica* (Septiembre 2004).

[BR12] Karel Bartos and Markus Rehak. Self-organized mechanism for distributed setup of multiple heterogeneous intrusion detection systems. In "Self-Adaptive and Self-Organizing Systems Workshops (SASOW), 2012 IEEE sixth international conference on", pages 31–38. IEEE (2012).

[Bru04] D. L. Bruening. Eficient, transparent, and comprehensive runtime code manipulation. *Ph.D., M.I.T. http://www.cag.lcs.mit.edu/dynamorio/* (2004).

[Bur05] M. Burgess. Probabilistic anomaly detection in distributed computer networks. *Science of Computer Programming, por publicar* (2005).

[BV00a] S. Bridges and R. Vaughn. Fuzzy data mining and genetic algorithms applied to intrusion detection. *Proceedings of the 2000 National Information Systems Security Conference (NISSC), Baltimore, USA* (Octubre 2000).

[BV00b] S. Bridges and R. Vaughn. Intrusion detection via fuzzy data mining. *Proceedings of the 12th Annual Canadian Information Technology Security Symposium* (Junio 2000).

[BV15] Igor Balabine and Alexander Velednitsky. Method and system for confident anomaly detection in computer network traffic (February 20 2015). US Patent App. 14/627,963.

[BWC02] D. Burroughs, L. Wilson, and G. Cybenko. Analysis of distributed intrusion detection systems using bayesian methods analysis of distributed intrusion detection systems using bayesian methods. *Proceedings of the 21th International Performance Computing and Communications Conference* (Abril 2002).

[C.02] Krügel C. Network alertness, towards an adaptative, collaborating intrusion detection system. *Dissertation for the degree of Doctor of Philosophy, University of Wien, Austria* (2002).

[CAT05] S. Chebrolu, A. Abraham, and J. Thomas. Feature deduction and ensemble design of intrusion detection systems. *Computers and Security Journal, Vol. 24/4, Pag. 295-307, Elsevier Science* (2005).

[CC02] S. N. Chari and P.-C. Cheng. Bluebox: A policy-driven, host-based intrusion detection system. *In Proceedings of the 2002 ISOC Symposium on Network and Distributed System Security (NDSS'02). San Diego, CA.* (2002).

[CC13] J Cheon and Tae-Young Choe. Distributed processing of snort alert log using hadoop. *Int J Eng Technol (0975-4024)* **5**(3), 2685–2690 (2013).

[CC98] Jannette Cardoso and Heloisa Camargo. "Fuzziness in Petri Nets, Physica-Verlag". (98).

[Cen15] Internet Storm Center. Reports - internet security (2015).

[CGH97] Enrique Castillo, José Manuel Gutiérrez, and Ali S Hadi. Sistemas expertos y modelos de redes probabilısticas. *Academia de Ingenierıa* (1997).

[Cha93] E. Charniak. Statistical languaje learning. *MIT Press, Cambridge, Massachusetts* (1993).

[CIS06] CISCO. Cisco intrusion detection home page. *Disponible en http://www.cisco.com/warp/public/cc/pd/sqsw/sqidsz/* (2006).

[CJ04] Mihai Christodorescu and Somesh Jha. Testing malware detectors. In "Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis", ISSTA '04, pages 34–44, New York, NY, USA (2004). ACM.

[CL04] Shapiro M. W. Cantrill, B. M. and A. H. Leventhal. Dynamic instrumentation of production systems. *6th Symposium on Operating Systems Design and Implementation,pp. 15?28.* (2004).

[cla15] claudemamo. kafka-web-console. a web console for apache kafka. (2015).

[CM98] J. Cannady and J. Mahaffey. The application of artificial intelligence to misuse detection. *Proceedings of the First Recent Advances in Intrusion Detection (RAID) Conference* (1998).

[CM00] A. Ghosh C. Michael. Using finite automata to mine execution data for intrusion detection: A preliminary report. *RAID 2000, LNCS 1907, pp.66-79* (2000).

[Cod90] Edgar F Codd. "The relational model for database management: version 2". Addison-Wesley Longman Publishing Co., Inc. (1990).

[Cox94] E. Cox. The fuzzy systems handbook: A practitioner's guide to building, using, and maintaining fuzzy systems. *AP Professional, Primera edició, ISBN 0121942708* (Febrero 1994).

[Cro02] T. Crothers. Implementing intrusion detection systems: A hands-on guide for securing the network. *John Whiley and Sons, Incorporated. ISBN 0764549499* (Diciembre 2002).

[Cro03] C. Kreibich; J. Crowcroft. Honeycomb, creating intrustion detection signatures using honeypots. *In 2nd Workshop on Hot Topics in Network* (2003).

[CS95] M. Crosbie and G. Safford. Applying genetic programming to intrusion detection. *Working Notes for the AAAI Symposium*

on Genetic Programming, Pag. 1-8, MIT, Cambridge, USA (Diciembre 1995).

[CSD+04] S. Chavan, K. Shah, N. Dave, S. Mukherjee, A. Abraham, and S. Sanyal. Adaptative neuro-fuzzy intrusion detection systems. *Proceedings of the 2004 International Conference on Information Technology: Coding and Computing (ITCC), Vol. 1, Pag. 70* (2004).

[CT09] Dawn M Cappelli and Randall F Trzeciak. Best practices for mitigating insider threat: Lessons learned from 250 cases. In "RSA Conferences" (2009).

[Cur94] T. W. Curry. Profiling and tracing dynamic library usage via interposition. *In USENIX Summer 1994 Technical Conference, pages 267 to 278, Boston, MA.* (1994).

[CW10] Hongmin Cai and Naiqi Wu. Design and implementation of a dids. In "2010 IEEE International Conference on Wireless Communications, Networking and Information Security", pages 340–342 (2010).

[CWM+07] Scott E Coull, Charles V Wright, Fabian Monrose, Michael P Collins, Michael K Reiter, et al.. Playing devil's advocate: Inferring sensitive information from anonymized network traces. In "NDSS", volume 7, pages 35–47 (2007).

[D.97] Upper D. Theory and algorithms for hidden markov models and generalized hidden markov models. *Dissertation for the degree of Doctor of Philosophy, Mathematics Department, University of California, Berkeley, USA* (Febrero 1997).

[D.01] Zamboni D. Using internal sensor for computer intrusion detection. *Center for Education and Research in Information Assurance and Security. Thesis submitted to the Faculty of Purdue University* (2001).

[D.02] Goldberg D. The design of innovation: Lessons from and for competent genetic algorithms. *Addison-Wesley, Primera edición, ISBN 1402070985* (Junio 2002).

[D03] González D. Sistemas de deteccion de intrusiones. *Version 1.01, Disponible en http://www.dgonzalez.net/pub/ids/* (Julio 2003).

[DCG12] Adrian J Duncan, Sadie Creese, and Michael Goldsmith. Insider attacks in cloud computing. *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications* **0**, 857–862 (2012).

[DD00] J.E. Dickerson and J.A. Dickerson. Fuzzy network profiling for intrusion detection. *Proceedings of the NAFIPS 19th International Conference of the North American Fuzzy Information Processing Society, Pag. 301-206, Atlanta, USA* (Julio 2000).

[DdIA98] FJ Díez and D de Inteligencia Artificial. Introducción al razonamiento aproximado. *Universidad de Educación a Distancia* (1998).

[DDW99] H. Debar, M. Dacier, and A. Wespi. Towards a taxonomy of intrusion-detection systems. *Computer Networks, Vol. 31, No. 8, Pag. 805-822* (Abril 1999).

[Dea01] D. Wagner; D. Dean. Intrusion detection via static analysis. *IEEE Symposium on Security and Privacy* (2001).

[Deb00] A. Wespi; M. Dacier; H. Debar. Intrusion detection using variable-length audit trail patterns. *Recent Advances in Intrusion Detection (RAID)* (2000).

[DEK+02] P. Dokas, L. Ertoz, V. Kumar, A. Lazarevic, J. Srivastava, and P. Tan. Data mining for network intrusion detection. *Proceedings of the NSF Workshop on Next Generation Data Mining, Baltimore, USA* (Noviembre 2002).

[Den87] D. Denning. An intrusion detection model. *IEEE Transactions on Software Engineering 13, 2 (Feb.), 222-232* (1987).

[Der14] Michael Derntl. Basics of research paper writing and publishing. *International Journal of Technology Enhanced Learning* **6**(2), 105–123 (2014).

[Dev15] Google Developers. Protocol buffers (2015).

[DGBJ14] Bruce Dang, Alexandre Gazet, Elias Bachaalany, and Sebastien Josse. "Practical Reverse Engineering". John Wiley & Sons Inc, Indianapolis, IN, edición: 1 edition (2014).

[DJKD01] J.E. Dickerson, J. Juslin, O. Koukousoula, and J.A. Dickerson. Fuzzy intrusion detection. *Proceedings of the IFSA World Congress and NAFIPS 20th International Conference of the North American Fuzzy Information Processing Society, Vol. 3, Pag. 1506-1510, Vancouver, Canada* (Julio 2001).

[DM11] P.K. Dick and M.E. Martín. "Ubik". Clásicos Minotauro. Grupo Planeta (2011).

[DMry] GIOVANNI VIGNA CHRISTOPHER KRUEGEL DARREN MUTZ, FREDRIK VALEUR. Anomalous system call detection. *University of California, Santa Barbara - Technical University of Vienna* (February).

[E.04] Lundin E. Logging for intrusion and fraud detection. *Thesis for the degree of Doctor of Philosophy, Department of Computer Engineering, Chalmers University, Göteborg, Suecia* (2004).

[E.12] Chickowski E. A case study in security big data analysis. Technical Report, DarkReading (2012).

[E.13] Chickowski E. Moving beyond siem for strong security analytics. Technical Report, DarkReading (2013).

[Eck01] S. Eckmann. Translating snort rules to statl scenarios. *Fourth International Symposium on Recent Advances in Intrusion Detection, University of California Davis, Lecture Notes in Computer Science 2212, Pag. 69-84* (Octubre 2001).

[EFB+15] Salma Elhag, Alberto Fernández, Abdullah Bawakid, Saleh Alshomrani, and Francisco Herrera. On the combination of genetic fuzzy systems and pairwise learning for improving detection rates on intrusion detection systems. *Expert Systems with Applications* **42**(1), 193–202 (2015).

[EGS00] E. Eskin, W. N. Grundy, and Y. Singer. Protein family classification using sparse markov transducers. *In Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology, Menlo Park, CA. AAAI Press.* (2000).

[ELSS01] Eleazar Eskin, Wenke Lee, J. Salvatore, and Stolfo. Modeling system calls for instrusion detecction with dynamic window sizes. (2001).

[Eri08] Jon Erickson. "Hacking: The Art of Exploitation". No Starch Press, San Francisco, CA, edición: 2 edition (February 2008).

[ES03] Agoston E Eiben and James E Smith. "Introduction to evolutionary computing". Springer Science & Business Media (2003).

[Ess10] Esset. Esset security report. Technical Report, Esset (2010).

[EV14] Mohsen Eslamnezhad and Ali Yazdian Varjani. Intrusion detection based on minmax k-means clustering. In "Telecommunications (IST), 2014 7th International Symposium on", pages 804–808. IEEE (2014).

[EVK00] S. Eckmann, G. Vigna, and R. Kemmerer. Statl: An attack languaje for state-based intrusion detection. *Technical Report, Department of Computer Science, University of California, Santa Barbara, USA* (2000).

[FBAF10] Romain Fontugne, Pierre Borgnat, Patrice Abry, and Kensuke Fukuda. Mawilab: combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking. In "Proceedings of the 6th International COnference", page 8. ACM (2010).

[FBH+10] BA Fessi, S Benabdallah, M Hamdi, S Rekhis, and N Boudriga. Data collection for information security system. In "Engineering Systems Management and Its Applications (ICESMA), 2010 second international conference on", pages 1–8. IEEE (2010).

[FHSL96a] S. Forrest, S. Hofmeyr, A. Somayaji, and T. Longstaff. A sense of self for unix processes. *Proceedings of the 1996 IEEE Symposium on Research in Security and Privacy, Pag. 120-128, IEEE Computer Society Press* (1996).

[FHSL96b] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A sense of self for unix processes. *In Proceedings of the 1996*

IEEE Symposium on Security and Privacy, pages , Los Alamitos, CA. IEEE Computer Society Press. (1996).

[fir15] FIRNSY. Github - barnyard2 (2015).

[FKFL03] H. FENG, O. KOLESNIKOV, P. FOGLA, AND W. LEE. Anomaly detection using call stack information. *In Proceedings of the 2003 IEEE Symposium on Security and Privacy.* (2003).

[Fon03] B. FONSECA. The ips question. *InfoWorld Magazine. Disponible en http://www.infoworld.com/article/03/04/04/14ips1.html?security* (Abril 2003).

[Fou15a] APACHE FOUNDATION. Apache spark - lightning-fast cluster computing. (2015).

[Fou15b] APACHE FOUNDATION. Mllib is apache spark's scalable machine learning library. (2015).

[Fou15c] THE APACHE SOFTWARE FOUNDATION. Apache avro is a data serialization system. (2015).

[Fou15d] THE APACHE SOFTWARE FOUNDATION. Apache crunch, simple and efficient mapreduce pipelines (2015).

[Fou15e] THE APACHE SOFTWARE FOUNDATION. Apache flume (2015).

[Fou15f] THE APACHE SOFTWARE FOUNDATION. Apache hbase (2015).

[Fou15g] THE APACHE SOFTWARE FOUNDATION. Apache hive tm (2015).

[Fou15h] THE APACHE SOFTWARE FOUNDATION. Apache kafka - a high-throughput distributed messaging system. (2015).

[Fou15i] THE APACHE SOFTWARE FOUNDATION. Apache sqoop (2015).

[Fou15j] THE APACHE SOFTWARE FOUNDATION. Apache storm (2015).

[Fou15k] THE APACHE SOFTWARE FOUNDATION. Apache zookeeper (2015).

[Fou15l] THE APACHE SOFTWARE FOUNDATION. Cassandra. (2015).

[Fou15m] THE APACHE SOFTWARE FOUNDATION. Welcome to apache hadoop (2015).

[Fou15n] The Apache Software Foundation. Welcome to apache pig (2015).

[Fra94] Jeremy Frank. Artificial intelligence and intrusion detection: Current and future directions. In "Proceedings of the 17th national computer security conference", volume 10, pages 1–12. Baltimore, USA (1994).

[Fyo97] Fyodor. The art of port scanning. *Disponible en http://www.insecure.org/nmap/* (1997).

[G.02] Wolf D. G. Statement before the house committee on government reform, subcommittee on government efficiency, financial management and intergovernmental relations, and the subcommittee for technology and procurement policy. *Federal Information Security Reform Act* (2002).

[G.13] Kaszuba G. packetloop/packetpig (2013).

[GA99] Schwartzbard A. y Schatz M. Gosh A. Learning program behavior profiles for intrusion detection. *Proceedings of the first USENIX Workshop on Intrusion Detection and Network Monitoring, Santa Clara, California, USA* (Abril 1999).

[GAH05] Crina Groşan, Ajith Abraham, and Sang Yong Han. Mepids: Multi-expression programming for intrusion detection system. In "Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach", pages 163–172. Springer (2005).

[Gar10] Gartner. Gartner security and risk management summit 2010. Technical Report, (2010).

[GBBS08] Abdoul Karim Ganame, Julien Bourgeois, Renaud Bidou, and Francois Spies. A global security architecture for intrusion detection on computer networks. *computers & security* **27**(1), 30–47 (2008).

[GC98] Wanken J. Ghosh, A. and F. Charron. Detecting anomalous and unknown intrusions against programs. *In Proceedings of the Annual Computer Security Application Conference (ACSAC'98). Scottsdale, AZ. 259-267* (1998).

[Ged10] Arvind Mewada; Shamaila Khan; Prafful Gedam. Network anomaly detection via clustering and custom kernel in msvm. *International Journal of Advanced Computer Science and Applications* **1**(1), 30 – 33 (Jan. 2010).

[GGL03] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In "ACM SIGOPS operating systems review", volume 37, pages 29–43. ACM (2003).

[GH99] D.B. G. Hunt. Detours: Binary interception of win32 functions. *Microsoft Research* (1999).

[GILB00] Angel Grediaga, Francisco Ibarra, Bernardo Ledesma, and Francisco Brotons. Utilización de redes neuronales para la detección de intrusos. *Departamento de Tecnología Informática y Computación. Universidad de Alicante, España* (2000).

[Gol03] Golomb. Intrusion detection methodologies demystified. *Enterasys Networks Incorporated Technical Whitepaper* (Diciembre 2003).

[Gol14] Vahid Golmah. An efficient hybrid intrusion detection system based on c5. 0 and svm. *International Journal of Database Theory and Application* **7**(2), 59–70 (2014).

[GR03] T. Garfinkel and M. Rosenblum. A virtual machine introspection based architecture for intrusion detection. *In Proceedings of the 2003 Network and Distributed System Security Symposium (NDSS)* (February, 2003).

[Gro15] Information Assurance Solutions Group. Defense in depth. Technical Report, National Security Agency (2015).

[GRS04a] Debin Gao, Michael K. Reiter, and Dawn Song. Gray-box extraction of execution graphs for anomaly detection. *Proceedings of the 11th ACM conference on Computer and communications security* (October 2004).

[GRS04b] Debin Gao, Michael K Reiter, and Dawn Song. Gray-box extraction of execution graphs for anomaly detection. In "Proceedings of the 11th ACM conference on Computer and communications security", pages 318–329. ACM (2004).

[GRY12] Manasi Gyanchandani, J Rana, and R Yadav. Taxonomy of anomaly based intrusion detection system: a review. *Neural Netw* **2**(43), 1–14 (2012).

[GW14] Meng Gao and Nihong Wang. A network intrusion detection method based on improved k-means algorithm. *Advanced Science and Technology Letters* **53**, 429–433 (2014).

[H.04a] Tijms H. Understanding probability: Chance rules in everyday life. *Cambrigde University Press, ISBN 0521540364* (Agosto 2004).

[H.04b] Zhang Z. Y Shen H. Online training of SVMs for real-time intrusion detection. *Proceedings of the International Conference on Advanced Information Networking and Applications (AINA), Vol. 1, Pag. 568* (2004).

[HB99] G. Hunt and D. Brubacher. Detours: Binary interception of win32 functions. *USENIX Windows NT Symposium* (1999).

[HD01] Mannila H. y Smyth P. Hand D. Principles of data mining. *MIT Press, ISBN 026208290X, Cambridge, Massachusetts, USA* (2001).

[HG03] Honavar V. Miller L. y Wang Y. Helmer G., Wong J. Lightweight agents for intrusion detection. *Elsevier Journal of Systems and Software, No. 67, Pag. 109-122* (2003).

[HJ92] R. Hastings and B. Joyce. Purify: Fast detection of memory leaks and access errors. *In Procedings of the Winter USENIX Technical Conference, San Francisco, CA.* (January 1992).

[HJWZ04] D. R. Simon H. J. Wang, C. Guo and A. Zugenmaier. Shield: Vulnerability-driven network filters for preventing known vulnerability exploits. *In Proceedings of the 2004 ACM SIGCOMM Conference* (August, 2004).

[HK03] Keromytis A. y Stolfo S. Heller K., Svore K. One class support vector machines for detecting anomalous windows registry accesses. *Proceedings of the ICDM Workshop on Data Mining for Computer Security (SMSEC), Melbourne, Florida, USA* (Noviembre 2003).

[HL90] Levitt K. N. Mukherjee B. Wood J. y Wolber D. Heberlein L.T., Dias G.V. A network security monitor. *Proceeding of the IEEE Symposium on Research in Security and Privacy, Pag. 296-304* (Mayo 1990).

[Ho01] S. Y. Ho. Intrusion detection systems for today and tomorrow. *SANS Institute. Information Security Reading Room* (2001).

[HSC+11] Shi-Jinn Horng, Ming-Yang Su, Yuan-Hsin Chen, Tzong-Wann Kao, Rong-Jian Chen, Jui-Lin Lai, and Citra Dwi Perkasa. A novel intrusion detection system based on hierarchical clustering and support vector machines. *Expert systems with Applications* **38**(1), 306–313 (2011).

[HT03] Meier M. y König H. Holz T. High-efficient intrusion detection infraestructure. *Infrastructure. DFN-Arbeitstagung über Kommunikationsnetze, 217-232* (2003).

[HW03] Liao Y. y Vemuri R. Hu W. Robust support vector machines for anomaly detection in computer security. *Proceedings of the 2003 International Conference on Machine Learning and Applications (ICMLA), Los Angeles, California, USA* (Junio 2003).

[IK95] Kemmerer R. y Porras P. Ilgun K. State transition analysis: A rule-based intrusion detection approach. *IEEE Transactions on Software Engineering, Vol. 21, No. 3, Pag. 181-199* (1995).

[IM03] Institute for Security ISECOM and Open Methodologies. Osstmm: Open source security testing methodology manual. (2003).

[Inc06a] Enterasys Networks Incorporated. Enterasys dragon intrusion defense. *Disponible en http://www.enterasys.com/products/ids/* (2006).

[Inc06b] Tripwire Incorporated. Tripwire change auditing solutions home page. *Disponible en http://www.tripwire.com/* (2006).

[Inc15] VMware Inc. Vmware (2015).

[Ins05] SANS Institute. SANS glossary of terms used in security and intrusion detection. *Disponible en http://www.sans.org/resources/glossary.php* (2005).

[ISG02] Y. Okazaki I. Sato and S. Goto. An improved intrusion detection method based on process profiling. *IPSJ Journal* (43), 3316 – 3326 (2002).

[J.88] Pearl J. Probabilistic reasoning in intelligent systems: Networks of plausible inference. *Morgan Kaufmann, ISBN 1558604790* (Septiembre 1988).

[J.99] Luo J. Integrating fuzzy logic with data mining methods for intrusion detection. *Technical Report, Mississippi State University* (1999).

[J.13] Oltsik J. Defining Big Data Security Analytics (April 2013).

[JBDCM00] B. Ravichandran J. B. D. Cabrera and R. K. Mehra. Statistical traffic modeling for network intrusion detection. *In Proceedings of the Eighth International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, pages 466-473, San Francisco, CA. IEEE Computer Society.* (August, 2000).

[JBDCM01] Lundy Lewis Joao B. D. Cabrera and Raman K. Mehra. Detection and classification of intrusions and faults using sequences of system calls. *Scientific Systems Company, 500 West Cummings Park, Suite 3000. Woburn MA 01801 USA* (December, 2001).

[JBT01] J. Desharnais M.M. Erhioui J. Bergeron, M. Debbabi and N. Tawbi. Static detection of malicious code in executable programs. (2001).

[JCR03] Scott M. Lewandowski Robert K. Cunningham Jesse C. Rabek, Roger I. Khazan. Detection of injected, dynamically generated, and obfuscated malicious code. *Massachusetts Institute of Technology, Lincoln Laboratory, 244 Wood Street, Lexington.* (October 2003).

[JD02] Klaus Julisch and Marc Dacier. Mining intrusion detection alarms for actionable knowledge. In "Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining", pages 366–375. ACM (2002).

[JHLY12] Hae-Duck J Jeong, WooSeok Hyun, Jiyoung Lim, and Ilsun You. Anomaly teletraffic intrusion detection systems on hadoop-based platforms: A survey of some problems and solutions. In "Network-Based Information Systems (NBiS), 2012 15th International Conference on", pages 766–770. IEEE (2012).

[Jon93] M. B. Jones. Interposition agents: Transparently interposing user code at the system interface. *In Proceedings of the Symposium on Operating Systems Principles, pages 80 to 93, Asheville, NC.* (1993).

[JPP11] V Jyothsna, VV Rama Prasad, and K Munivara Prasad. A review of anomaly based intrusion detection systems. *International Journal of Computer Applications* **28**(7), 26–35 (2011).

[JTGM02] S. Jha J. T. Giffin and B. P. Miller. Detecting manipulated remote call streams. *USENIX Security Symposium.* (August 2002).

[JTGM04] S. Jha J. T. Giffin and B. P. Miller. Efficient context-sensitive intrusion detection. *11th Annual Network and Distributed System Security Symposium.* (February 2004.)

[JV91] H. S. Javitz and A. Valdes. The sri ides statistical anomaly detector. *In Proceedings of the IEEE Symposium on Security and Privacy* (1991).

[K.93] Ilgun K. Ustat: A real-time intrusion detection system for unix. *Proceedings of the IEEE Symposium on Research on Security and Privacy, Oakland, USA* (Mayo 1993).

[K.99] Jackson K. Intrusion detection system product survey. *Technical Report LA-UR-99-3883. Distributed Knowledge Systems Team, Computer Research and Applications Group, Los Alamos National Laboratory, New Mexico* (Junio 1999).

[K.01a] Murphy K. The bayes net toolbox for matlab. *Computing Science and Statistics, Vol. 33* (Octubre 2001).

[K.01b] Valdés A. Y Skiner K. Probabilistic alert correlation. *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection, Pag. 54-68* (2001).

[K.03] Julisch K. Using root cause analysis to handle intrusion detection alarms. *IBM Zurich Research Laboratory. Dissertation for the degree of Doctor of Philosophy, University of Dortmund, Alemania* (2003).

[K.14] Marko K. Big data: Cyber security silver bullet? intel makes the case. Technical Report, Forbes (2014).

[KC03a] C. Kreibich and J. Crowcroft. Honeycomb - creating intrusion detection signatures using honeypots. *In Proceedings of the Second Workshop on Hot Topics in Networks (HotNets-II)* (November, 2003).

[KC03b] Robertson W. y Valeur F. Krügel C., Mutz D. Bayesian event classification for intrusion detection. *Proceedings of the 19th Annual Computer Security Applications Conference, Las Vegas* (Diciembre 2003).

[KC12] Igor Kotenko and Andrey Chechulin. Common framework for attack modeling and security evaluation in siem systems. In "Green Computing and Communications (GreenCom), 2012 IEEE international conference on", pages 94–101. IEEE (2012).

[KD05] Nguyen H. y Park J. Kim D. Genetic algorithm to improve svm based network intrusion detection system. *Proceedings of the 19th International Conference on Advanced Information Networking and Applications (AINA), Vol. 2, Pag. 155-158* (2005).

[KK04] Hyang-Ah Kim and Brad Karp. Autograph: Toward automated, distributed worm signature detection. In "USENIX security symposium", volume 286. San Diego, CA (2004).

[KMK15] Oliver Rochford Kelly M. Kavanagh. Magic quadrant for security information and event management. Technical Report, Gartner (2015).

[Kon11] Jinzhu Kong. Adjointvm: a new intrusion detection model for cloud computing. *Energy Procedia* **13**, 7902–7911 (2011).

[KPS12] Igor Kotenko, Olga Polubelova, and Igor Saenko. The ontological approach for siem data repository implementation. In "Green Computing and Communications (GreenCom), 2012 IEEE international conference on", pages 761–766. IEEE (2012).

[KRL97] Calvin Ko, Manfred Ruschitzka, and Karl Levitt. Execution monitoring of security-critical programs in distributed systems: A specification-based approach. In "Security and Privacy, 1997. Proceedings., 1997 IEEE Symposium on", pages 175–187. IEEE (1997).

[Kro14] Kroll. 2013/2014 Global Fraud Report: Who's Got Something to Hide? page 48 (2014).

[L.98] Me L. Gassata, a genetic algorithm as an alternative tool for security audit trails analysis. *First International Workshop on the Recent Advances in Intrusion Detection, Louvain-la-Neuve, Belgica* (Septiembre 1998).

[LA03] Kumar V. Ozgur A. y Srivastava J. Lazarevic A., Ertoz L. A comparative study of anomaly detection schemes in network intrusion detection. *SIAM International Conference on Data Mining* (Mayo 2003).

[Lan01] Doug Laney. 3d data management: Controlling data volume, velocity and variety. *META Group Research Note* **6**, 70 (2001).

[LB05] Zhen Liu and Susan M. Bridges. Dynamic learning of automata from the call stack log for anomaly detection. *Department of Computer Science and Engineering. Mississippi State University* (April, 2005).

[LC04] Lap Chung Lam and Tzi-cker Chiueh. Automatic extraction of accurate application-specific sandboxing policy. In "Recent Advances in Intrusion Detection", pages 1–20. Springer (2004).

[LCG10] Fang Lan, Wang Chunlei, and Ma Guoqing. A framework for network security situation awareness based on knowledge discovery. In "Computer Engineering and Technology (ICCET),

2010 2nd international conference on", volume 1, pages V1–226. IEEE (2010).

[Lev97] C. Ko ; M. Ruschitzka; K. Levitt. Execution monitoring of security-critical programs in distributed systems: A specification-based approach. *In Proceedings of the 1997 IEEE Symposium on Security and Privacy* (1997).

[LF09] Gordon Lyon and Fyodor. "Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning". Nmap Project, Sunnyvale, CA (2009).

[LH05] Cohn R. Muth R. Patil H. Klauser A. Lowney G. Wallace S. Reddi V. J. Luk, C. and K. Hazelwoo. Pin: Building customized program analysis tools with dynamic instrumentation. *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), Chicago, IL, USA, pp. 190?200.* (2005).

[Lin15] LinkedIn. Linkedin (2015).

[LKS05] Aleksandar Lazarevic, Vipin Kumar, and Jaideep Srivastava. Intrusion detection: A survey. In "Managing Cyber Threats", pages 19–78. Springer (2005).

[LKS10] Youngseok Lee, Wonchul Kang, and Hyeongu Son. An Internet traffic analysis method with MapReduce. *2010 IEEE/IFIP Network Operations and Management Symposium Workshops* pages 357–361 (2010).

[LL13] Yeonhee Lee and Youngseok Lee. Toward scalable internet traffic measurement and analysis with hadoop. *ACM SIGCOMM Computer Communication Review* **43**(1), 5–13 (2013).

[LLC06] Lap Chung Lam, Wei Li, and Tzi-cker Chiueh. Accurate and automated system call policy-based intrusion prevention. In "Dependable Systems and Networks, 2006. DSN 2006. International Conference on", pages 413–424. IEEE (2006).

[LP99] U. Lindqvist and P. Porras. Detecting computer and network misuse with the production based expert system toolset (p-best). *In IEEE Symposium on Security and Privacy. Oakland, CA. 146-161.* (1999).

[LS95] J. Larus and E. Schnarr. Eel: Machine-independent executable editing. *In Proceedings of the ACM SIGPLAN '95 Conference on Programming Language Design and Implementation, La Jolla, CA.* (June 1995).

[LS98] W. Lee and S. Stolfo. Data mining approaches for intrusion detection. *In Proceedings of the 7th USENIX Security Symposium* (1998).

[LSC+01] Wenke Lee, Salvatore J Stolfo, Philip K Chan, Eleazar Eskin, Wei Fan, Matthew Miller, Shlomo Hershkop, and Junxin Zhang. Real time data mining-based intrusion detection. In "DARPA Information Survivability Conference & Exposition II, 2001. DISCEX'01. Proceedings", volume 1, pages 89–100. IEEE (2001).

[LT90] Gilham F. Jagannathan R. Newumann P. y Jalali C. Lunt T., Tamaru A. Ides: A progress report. *Proceedings of the 6th Annual Computer Security Applications Conference* (1990).

[Lun93] T. F. Lunt. Detecting intruders in computer systems. *In Proceedings of the Sixth Annual Symposium and Technical Displays on Physical and Electronic Security*, (1993).

[LW99] Stolfo S. y Mok K. Lee W. A data mining framework for building intrusion detection models. *Proceedings of the IEEE Symposium on Security and Privacy, Pag. 120-132* (1999).

[LY01] Tseng S. y Lin S. Lin Y. An intrusion detection model based upon intrusion detection markup languaje (idml). *Department of Computer and Information Science, National Chiao Tung University, Taiwan* (Agosto 2001).

[M.00] Ye N. Y Xu M. Probabilistic networks with undirected links for anomaly detection. *Workshop on Information Assurance and Security, Proceedings of the IEEE, Pag. 175-179* (2000).

[M.01a] Ranum M. Coverage in intrusion detection systems. *Technical Publication. NFR Security Incorporated Technical Publication* (Junio 2001).

[M.01b] Ranum M. Experiences benchmarking intrusion detection systems. *NFR Security Incorporated Technical Publication* (Diciembre 2001).

[M.03a] Llorens M. Redes reconfigurables. modelizacion y verificacion. *Tesis Doctoral, Departamento de Sistemas Informaticos y Computacion, Universidad Politecnica de Valencia* (2003).

[M.03b] Mahoney M. Network traffic anomaly detection based on packet bytes. *Proceedings of the ACM Symposium on Applied Computing, Pag. 346-350, Melbourne, Florida, USA* (2003).

[M.12] Rouse M. Security information and event management (2012).

[Mar01] Carla Marceau. Characterizing the behavior of a program using multiple-length n-grams. *Odyssey Research Associates. Ithaca, NY 14850* (February, 2001).

[MB00] Luigi V. Mancini Massimo Bernaschi, Emanuele Gabrielli. Operating system enhancements to prevent the misuse of system calls. *IAC-CNR. Viale del Policlinico, 137. 00161 Rome, Italy* (2000).

[MB04] Heberlein T. y Levitt K. Mukherjee B. Network intrusion detection. *IEEE Network, 8(3):26-41* (Junio 2004).

[MC03] Matthew V Mahoney and Philip K Chan. An analysis of the 1999 darpa/lincoln laboratory evaluation data for network anomaly detection. In "Recent Advances in Intrusion Detection", pages 220–237. Springer (2003).

[McH00] John McHugh. Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM transactions on Information and system Security* **3**(4), 262–294 (2000).

[MDV13] Sachin Meena, Esther Daniel, and NA Vasanthi. Survey on various data integrity attacks in cloud environment and the solutions. In "Circuits, Power and Computing Technologies (ICCPCT), 2013 International Conference on", pages 1076–1081. IEEE (2013).

[MG11] Peter M. Mell and Timothy Grance. Sp 800-145. the nist definition of cloud computing. Technical Report, Gaithersburg, MD, United States (2011).

[MGM11] Justin Myers, Michael R Grimaila, and Robert F Mills. Log-based distributed security event detection using simple event correlator. In "System Sciences (HICSS), 2011 44th Hawaii International Conference on", pages 1–7. IEEE (2011).

[MHR11] Stefan Metzger, Wolfgang Hommel, and Helmut Reiser. Integrated security incident management–concepts and real-world experiences. In "IT Security Incident Management and IT Forensics (IMF), 2011 Sixth International Conference On", pages 107–121. IEEE (2011).

[Mic15] Trend Micro. Ossec - open source security (2015).

[Mil02] J. T. Giffin; S. Jha; B. Miller. Detecting manipulated remote call streams. *11th USENIX Security Symposium* (2002).

[Mit12] Harvey B Mitchell. "Data Fusion: Concepts and Ideas: Concepts and Ideas". Springer Science & Business Media (2012).

[MJ00] Christie A. y Allen J. McHugh J. Defending yourself: The role of intrusion detection systems. *IEEE Software, Pag. 42-51* (Octubre 2000).

[MJ+14] Leandros Maglaras, Jianmin Jiang, et al.. Ocsvm model combined with k-means recursive clustering for intrusion detection in scada systems. In "Heterogeneous Networking for Quality, Reliability, Security and Robustness (QShine), 2014 10th International Conference on", pages 133–134. IEEE (2014).

[Mos07] C.; Kirda E. Moser, A.; Kruegel. Limists of static malware detection. *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual* pages 421–430 (Dec. 2007).

[MP05] Wes Masri and Andy Podgurski. Using dynamic information flow analysis to detect attacks against applications. *Computer Science Department. American University of Beirut. Beirut,*

Lebanon 1107 2020 - Electrical Engineering and Computer Science Department. Case Western Reserve University. Cleveland, OH 44106 (May, 2005).

[MPB+13] Chirag Modi, Dhiren Patel, Bhavesh Borisaniya, Hiren Patel, Avi Patel, and Muttukrishnan Rajarajan. A survey of intrusion detection techniques in cloud. *Journal of Network and Computer Applications* **36**(1), 42–57 (2013).

[MR06] Trevor Jim Richard Schlichting Mohan Rajagopalan, Matti Hiltunen. Authenticated system calls. *Department of Computer Science, The University of Arizona. AT and T Labs-Research, 180 Park Avenue* (July, 2006).

[MRR12] Amuthan Prabakar Muniyandi, R Rajeswari, and R Rajaram. Network anomaly detection by cascading k-means clustering and c4. 5 decision tree algorithm. *Procedia Engineering* **30**, 174–182 (2012).

[MS05] Sung A. y Abraham A. Mukkamala S. Intrusion detection using an ensemble of intelligent paradigms. *Journal of Network and Computer Applications, No. 28, Pag. 167-182, Elsevier Science* (2005).

[MS06] Kevin D. Mitnick and William L. Simon. "The Art of Intrusion: The Real Stories Behind the Exploits of Hackers, Intruders & Deceivers: The Real Stories Behind the Exploits of Hackers, Intruders and Deceivers". John Wiley & Sons, Indianapolis, IN, edición: new ed edition (2006).

[MYSU14] Z Muda, W Yassin, MN Sulaiman, and NI Udzir. K-means clustering and naive bayes classification for intrusion detection. *Journal of IT in Asia* **4**, 14 (2014).

[MZ07] Federico Maggi and Stefano Zanero. On the use of different statistical tests for alert correlation: short paper. In "Proceedings of the 10th international conference on Recent advances in intrusion detection", RAID'07, pages 167–177, Berlin, Heidelberg (2007). Springer-Verlag.

[NaBM13] Mohamed Nassar, Bechara al Bouna, and Qutaibah Malluhi. Secure outsourcing of network flow data analysis.

In "Big Data (BigData Congress), 2013 IEEE International Congress On", pages 431–432. IEEE (2013).

[NC15] B Natarajan and P Chellammal. Enhanced k-means++ clustering for big data with mapreduce. (2015).

[NKS05] James Newsome, Brad Karp, and Dawn Song. Polygraph: Automatically generating signatures for polymorphic worms. In "Security and Privacy, 2005 IEEE Symposium on", pages 226–241. IEEE (2005).

[NP99] P. G. Neumann and P. A. Porras. Experience with emerald to date. *In Proceedings of the Usenix Workshop on Intrusion Detection* (1999).

[NS03] N. Nethercote and J. Seward. Valgrind: A program supervision framework. *In 3rd Workshop on Runtime Verification* (2003).

[NS05] James Newsome and Dawn Song. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. (2005).

[NSA06] NSA. National security agency home page. *Disponible en http://www.nsa.gov/* (2006).

[OAS13] Ucman Oktay, Muhammed Ali Aydin, and Ozgur Koray Sahingoz. Circular chain vm protection in adjointvm. In "Technological Advances in Electrical, Electronics and Computer Engineering (TAEECE), 2013 International Conference on", pages 93–97. IEEE (2013).

[OKS+04] David Ogle, Heather Kreger, Abdi Salahshour, Jason Cornpropst, Eric Labadie, Mandy Chessell, Bill Horn, John Gerken, James Schoech, and Mike Wamboldt. Canonical situation data format: The common base event v1. 0.1. *IBM Corporation* (2004).

[P.92] Winston P. Artificial intelligence. *Addison Wesley, Reading, Massachusetts, Tercera Edición, ISBN 0201533774* (Enero 1992).

[P.96] Proctor P. Computer misuse detection system (cmds) concepts. *Science Applications International Corporation (SAIC), Technical Paper* (1996).

[P.01] Inella P. The evolution of intrusion detection systems. *2001. Security Focus. Disponible en http://www.securityfocus.com/infocus/1514* (2001).

[PAGT07] Sandhya Peddabachigari, Ajith Abraham, Crina Grosan, and Johnson Thomas. Modeling intrusion detection system using hybrid intelligent systems. *Journal of network and computer applications* **30**(1), 114–132 (2007).

[Pax98] V. Paxson. Bro: A system for detecting network intruders in real-time. *In Proceedings of 7th USENIX Security Symposium. San Antonio, TX* (1998).

[PBW03] K. Fraser S. Hand T. Harris A. Ho R. Neugebauer I. Pratt P. Barham, B. Dragovic and A. Warfield. Xen and the art of virtualization. *In Proceedings of the 2003 Symposium on Operating Systems Principles* (October, 2003).

[PCJD08] Mila Dalla Preda, Mihai Christodorescu, Somesh Jha, and Saumya Debray. A semantics-based approach to malware detection. *ACM Trans. Program. Lang. Syst.* **30**, 25:1–25:54 (September 2008).

[Pic78] J. Picciotto. The desing of an effective auditing subsystem. *The MITRE Corporation, Bedford, Massachusetts* (1978).

[PM04] Eloff J. y Venter H. Pillai M. An approach to implement a network intrusion detection system using genetic algorithms. *Proceedings of the 2004 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries, ACM International Conference Proceedings Series, Vol. 75* (2004).

[Poi15] United States Military Academy West Point. Data sets. Internet (2015).

[PP07] Animesh Patcha and Jung-Min Park. An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer networks* **51**(12), 3448–3470 (2007).

[PPWP15] Supriya Pharande, Priyanka Pawar, PW Wani, and AB Patki. Application of hurst parameter and fuzzy logic for denial of service attack detection. In "Advance Computing Conference (IACC), 2015 IEEE International", pages 834–838. IEEE (2015).

[PS05] Grosan C. y Thomas J. Peddabachigari S., Abraham A. Modeling intrusion detection system using hybrid intelligent systems. *Journal of Network and Computer Applications, Elsevier Science* (2005).

[PS15] Asry Faidhul Ashaari Pinem and Erwin Budi Setiawan. Implementation of classification and regression tree (cart) and fuzzy logic algorithm for intrusion detection system. In "Information and Communication Technology (ICoICT), 2015 3rd International Conference on", pages 266–271. IEEE (2015).

[PSMP15] Muhammad Syafiq Mohd Pozi, Md Nasir Sulaiman, Norwati Mustapha, and Thinagaran Perumal. Improving anomalous rare attack detection rate for intrusion detection system using support vector machine and genetic programming. *Neural Processing Letters* pages 1–12 (2015).

[PUR05] Uday Joshi Prem Uppuluri and Arnab Ray. Preventing race condition attacks on file-systems. *Dept. of Computer Science University of Missouri at Kansas City, MO, 641 10 - Dept. of Computer Science State University of New York Stony Brook, NY, 11790* (2005).

[R+99] Martin Roesch et al.. Snort: Lightweight intrusion detection for networks. In "LISA", volume 99, pages 229–238 (1999).

[R.00] Shirey R. Internet security glossary. *Internet RFC 2828. The Internet Society, Network Working Group, GTE/BBN Technologies. Disponible en http://www.ietf.org/rfc/ /rfc2828.txt* (Mayo 2000).

[Ran15] Meesala Shobha Rani. A Survey on Taxonomy of Intrusion Detection System (IDS) and Anomaly Based Intrusion Detection System (ABIDS) Techniques. *International Journal of Advance Research in Computer Science and Management Studies* **3**(6), 128–136 (2015).

[Rap15] Rapid7. Metasplot - world's most used penetration testing sotfware. (2015).

[Red15] RedBorder. Barnyard2 with json to kafka output plugin (2015).

[Res15] Mosaic Security Research (2015).

[RG05a] Jan Vitek Rajeev Gopalakrishna, Eugene H. Spafford. Efficient intrusion detection using automaton inlining. *Center for Education and Research in Information Assurance and Security. Department of Computer Sciences. Purdue University* (May, 2005).

[RG05b] Culbert C. Donell B. Ly B. Ortiz C. Giarrantano J. y López F. Riley G., Savely R. Clips: A tool for building expert systems. *Disponible en http://www.ghg.net/clips/CLIPS.html* (2005).

[RIAH11] John C Roberts II and Wasim Al-Hamdani. Who can you trust in the cloud?: a review of security issues within cloud computing. In "Proceedings of the 2011 Information Security Curriculum Development Conference", pages 15–19. ACM (2011).

[RJ86] L. R. Rabiner and B. H. Juang. An introduction to hidden markov models. *IEEE ASSP Magazine, pages 4 to 16, Jannuary.* (1986).

[RK11] Shirley Radack and Rick Kuhn. Managing security: The security content automation protocol. *IT professional* **13**(1), 9–11 (2011).

[RKMJ00] J. Gagnon R. K. Mukkamala and S. Jajodia. Integrating data mining techniques with intrusion detection. *In V. Atluri and J. Hale, editors, Research Advances in Database and Information Systems Security, pages 33-46. Kluwer Publishers* (2000).

[RM97] Stolarchuk M. Sienkiewicz M. Lambeth A. y Wall E. Ranum M., Landfield K. Implementing a generalized tool for network monitoring. *Proceedings of the 11th Systems Administration Conference, San Diego, California, USA* (Octubre 1997).

[Roy06] M.; Dagon D.; Edmonds R.; Wenke Lee; Royal, P.; Halpin. Polyunpack: Automating the hidden-code extraction of unpack-executing malware. *Computer Security Applications Conference, 2006. ACSAC '06. 22nd Annual* pages 289 – 300 (Dec. 2006).

[RS15] Maryam Rostamipour and Babak Sadeghiyan. An architecture for host-based intrusion detection systems using fuzzy logic. *Journal of Network and Information Security* 2(2) (2015).

[RSD01] P. Bollineni R. Sekar, M. Bendre and D. Dhurjati. A fast automaton-based method for detecting anomalous program behaviors. *IEEE Symposium on Security and Privacy, pages 144 to 155.* (2001).

[RSZ02] J. Frullo T. Shanbhag A. Tiwari H. Yang R. Sekar, A. Gupta and S. Zhou. Specification-based anomaly detection: A new approach for detectin network intrusions. *ACM Computer and Communication Security Conference* (2002).

[S.12] Laplace P. S. Théorie analytique des probabilités. *Courcier Imprimeur, Paris* (1812).

[S.88] Smaha S. Haystack: An intrusion detection system. *Proceedings of the 4th Aerospace Computer Security Applications Conference, Orlando, FL* (Diciembre 1988).

[S.95] Kumar S. Classification and detection of computer intrusions. *Thesis submitted for the degree of Doctor of Philosophy, Purdue University, USA* (Agosto 1995).

[S.96] Smaha S. Y Snapp S. Method and system for detecting intrusion into and misuse of a data processing system. *US 555742, US Patent Office* (Septiembre 1996).

[S.02] Gould S. The structure of evolutionary theory. *Belknap Press, Primera edición, ISBN 0674006135* (Marzo 2002).

[S.04] Scott S. A bayesian paradigm for designing intrusion detection systems. *Computational Statistics and Data Analysis, Elsevier Science, Vol. 45, Issue. 1, Pag. 69-83* (Febrero 2004).

[SAHS98] S. Forrest S. A. Hofmeyr and A. Somayaji. Intrusion detecction using sequences of system calls. *Journal of Computer Security, 6:151 to 180.* (1998).

[SC99] Pierce L. Y Matzner S. Sinclair C. An application of machine learning to network intrusion. *Proccedings of the 15th Annual Computer Security Applications Conference (ACSAC), Pag. 371* (1999).

[SC05] A. Smirnov and T. Chiueh. Dira: Automatic detection, identification, and repair of control-hijacking attacks. *In Proceedings of The 12th Annual Network and Distributed System Security Symposium (NDSS '05)* (February, 2005).

[Sca15] Scala. Object-oriented meets functional (2015).

[SE94] A. Srivastava and A. Eustace. Atom - a system for building customized program analysis tools. *In Proceesings of the Symposium on Programming Language Design and Implementation, pages 196 to 205, Orlando, FL.* (1994).

[Sec15] Offensive Security. Kali linux - cutting edge penetration testing tools (2015).

[Sen15] SentinelOne. Advanced Threat Intelligence Report 2015 Predictions. Technical Report, (2015).

[SEVS04] Sumeet Singh, Cristian Estan, George Varghese, and Stefan Savage. Automated worm fingerprinting. In "OSDI", volume 4, pages 4–4 (2004).

[SF94] L. Allen R. Cherukuri S. Forrest, A.S. Perelson. Self-nonself discrimination in a computer. *IEEE Symposium on Security and Privacy* (1994).

[SH98] A. Somayaji S. Hofmeyr, S. Forrest. Intrusion detection using sequences of system calls. *Journal of Computer Security, vol. 6, pp. 151-180* (1998).

[shi15] shirkdog. Github project - pulledpork (2015).

[SHS98] S. Forrest S. Hofmeyr and A. Somayaji. Intrusion detection using sequences of system calls. *Journal of Computer Security* pages 151–180 (1998).

[SJ14] Santosh Kumar Sahu and Sanjay Kumar Jena. A study of k-means and c-means clustering algorithms for intrusion detection product development. (2014).

[SKRC10] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In "Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on", pages 1–10. IEEE (2010).

[SM88] Hanna M. Y Whitehurst R. Sebring M., Shellhouse E. Expert systems in intrusion detection, a case study. *Proceedings of the 11th National Computer Security Conference, Baltimore, USA* (1988).

[SM12] Neelam Sharma and Saurabh Mukherjee. A novel multi-classifier layered approach to improve minority attack detection in ids. *Procedia Technology* **6**, 913–921 (2012).

[Sno05] Snort. Snort: The facto standard for intrusion Detection/Prevention home page. *Disponible en http://www.snort.org/* (2005).

[Sof15] Tableau Software. Tableau - fast analytics for everyone. (2015).

[SP10] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. In "Security and Privacy (SP), 2010 IEEE Symposium on", pages 305–316. IEEE (2010).

[SP12] Shan Suthaharan and Tejaswi Panchagnula. Relevance feature selection with data cleaning for intrusion detection system. In "Southeastcon, 2012 Proceedings of IEEE", pages 1–6. IEEE (2012).

[Spa15] Branko Spasojevic. "Gray Hat Hacking The Ethical Hacker's Handbook, Fourth Edition". McGraw-Hill Osborne, s.l, edición: 4 edition (February 2015).

[Spl15] Splunk (2015).

[SS91] Dias G. Goan T. Heberlein T. Ho C. Levitt K. Mukherjee B. Smaha S. Grance T. Teal D. Y Mansur D. Snapp S.,

Brentano J. DIDS (distributed intrusion detection system) - motivation, architecture, and an early prototype. *Proceedings of the 14th National Computer Security Conference, Washington, DC* (Octubre 1991).

[SSK05] S.W. Boyd S. Sidiroglou, M.E. Locasto and A.D. Keromytis. Building a reactive immune system for software services. *In Proceedings of USENIX Annual Technical Conference, pages 149 - 161* (April, 2005).

[SSR+12] Madhan Kumar Srinivasan, K Sarukesi, Paul Rodrigues, M Sai Manoj, and P Revathy. State-of-the-art cloud computing security taxonomies: a classification of security challenges in the present cloud computing environment. In "Proceedings of the international conference on advances in computing, communications and informatics", pages 470–476. ACM (2012).

[SSTG12] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali A Ghorbani. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers & Security* **31**(3), 357–374 (2012).

[SSZ+13] Dinkar Sitaram, Manish Sharma, Mariyah Zain, Ankita Sastry, and Rishika Todi. Intrusion detection system for high volume and high velocity packet streams: A clustering approach. *Int J Innovation Manag Technol* **4**(5), 480–485 (2013).

[Sto04] K. Wang; S. J. Stolfo. Anomalous payload-based network intrusion detection. *In Proceedings of the 7th International Symposium on (RAID)* pages 201–222 (2004).

[STO+11] Jungsuk Song, Hiroki Takakura, Yasuo Okabe, Masashi Eto, Daisuke Inoue, and Koji Nakao. Statistical analysis of honeypot data and building of kyoto 2006+ dataset for nids evaluation. In "Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security", pages 29–36. ACM (2011).

[Su11] Ming-Yang Su. Real-time anomaly detection systems for denial-of-service attacks by weighted k-nearest-neighbor clas-

sifiers. *Expert Systems with Applications* **38**(4), 3492–3498 (2011).

[sur10] Suricata - open source ids / ips / nsm engine (January 2010).

[SVK15] B Senthilnayaki, K Venkatalakshmi, and A Kannan. Intrusion detection using optimal genetic feature selection and svm based classifier. In "Signal Processing, Communication and Networking (ICSCN), 2015 3rd International Conference on", pages 1–4. IEEE (2015).

[Sym13] Symantec Corporation. Internet Security Threat Report. Technical Report April, (2013).

[T.00] Lane T. Machine learning techniques for the computer security domain of anomaly detection. *Thesis submitted for the degree of Doctor of Philosophy, Purdue University, USA* (Agosto 2000).

[T.02] Goeldenitz T. Ids today and tomorrow. *SANS Institute Information Security Reading Room* (Enero 2002).

[T.03a] Toth T. Improving intrusion detection systems. *Dissertation submitted for the degree of Doctor of Philosophy, University of Vienna, Austria* (2003).

[T.03b] Wickham T. Intrusion detection is dead. long live to intrusion prevention! *SANS Institute Information Security GIAC Certification Practical, version GSEC 1.4b* (Abril 2003).

[T.04] Holland T. Understanding ips and ids: Using ips and ids together for defense in depth. *SANS Institute Information Security Reading Room* (Febrero 2004).

[TAF01] C. Taylor and J. Alves-Foss. Nate, network analysis of anomalous traffic events, a low-cost approach. *New Security Paradigms Workshop* (2001).

[TD01] William MK Trochim and James P Donnelly. Research methods knowledge base. (2001).

[Tea03] CERT/In Indian Computer Emergency Response Team. Ids - intrusion detection system. *CERT/In Security Guideline CISG-2003-06* (2003).

[Tec15] UBM Tech. Blackhat (2015).

[TH90] Chen K. Y Lu S. Teng H. Adaptative real-time anomaly detection using inductively generated sequential patterns. *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy, Oakland, USA* (Mayo 1990).

[TKM02] Kymie MC Tan, Kevin S Killourhy, and Roy A Maxion. Undermining an anomaly-based intrusion detection system using common exploits. In "Recent Advances in Intrusion Detection", pages 54–73. Springer (2002).

[TL97] C.E. Brodley T. Lane. Sequence matching and learning in anomaly detection for computer security. *AAAI Workshop: AI Approaches to Fraud Detection and Risk Management, pp.49-49* (1997).

[TL99] C.E. Brodley T. Lane. Temporal sequence learning and data reduction for anomaly detection. *ACM Trans. Information and System Security, vol. 2, no. 3, pp. 295-331* (1999).

[TL01] D. Thain and M. Livny. Multiple bypass: Interposition agents for distributed computing. *Journal of Cluster Computing, 4(1):39 to 47.* (March 2001).

[TM02a] K. Tan and R. Maxion. Why 6?? defining the operational limits of stide, an anomaly based intrusion detector. *In Proceedings of the IEEE Symposium on Security and Privacy. Oakland, CA. 188-202* (2002).

[TM02b] Killourhy K. Tan, K. and R. Maxion. Undermining an anomaly-based intrusion detection system using common exploits. *In Proceedings of RAID 2002. Zurich, CH.* (2002).

[tns15] tenable network security. Nessus (2015).

[TP05] Steinbach M. Y Kumar V. Tan P. Introduction to data mining. *Addison Wesley, Primera Edición, ISBN 0321321367* (Mayo 2005).

[TRB97] D. Lee A. Wolman W. Wong H. Levy T. Romer, G. Voelker and B. Bershad. Instrumentation and optimization of

win32/intel executables using etch. *In USENIX Windows NT Workshop, Seattle, WA.* (1997).

[Tre15] A Trendlabs. THE INVISIBLE BECOMES Trend Micro Security Predictions for 2015 and Beyond. Technical Report, (2015).

[Tri] Tripwire. Top 5 malware trends on the horizon.

[Tri06] Open Source Tripwire. Open source tripwire home page. *Disponible en http://sourceforge.net/projects/tripwire/* (2006).

[Tri08] Pierre Chifflier; Sébastien Tricaud. Intrusion detection systems correlation. *CanSecWest Vancouver* (2008).

[U.99] Lindqvist U. On the fundamentals of analysis and detection of computer misuse. *Thesis for the degree of Doctor of Philosophy, Chalmers University of Technology. Göteborg, Suecia* (1999).

[Uni15] Stanford University. Stanford encyclopedia of philosophy. *Consulta del término "Bayes'Theorem". Disponible en http://plato.stanford.edu/entries/bayes-theorem/* (2015).

[USR+12] Mathias Uslar, Michael Specht, Sebastian Rohjans, Jörn Trefke, and José M González. "The Common Information Model CIM: IEC 61968/61970 and 62325-A practical introduction to the CIM", volume 66. Springer Science & Business Media (2012).

[V.95] Vapnik V. The nature of statistical learning theory. *Springer-Verlag, ISBN 0387945598* (1995).

[VG00] Eckman S. Y Kemmerer R. Vigna G. The STAT tool suite. *Proceedings of DISCEX 2000, IEEE Press, Hilton Head, South Carolina* (Enero 2000).

[VG13] SANJAI VeetiL and QIGANG Gao. A real-time intrusion detection system by integrating hadoop and naive bayes classification. In "Dalhousie Computer Science In-house Conference (DCSI)" (2013).

[VP01] Mahoney M. V. and Chan P.K. Learning rules for anomaly detection of hostile network traffic. (2001).

[W.04] Li W. Using genetic algorithm for network intrusion detection. *Proceedings of the United States Department of Energy Cyber Security Group 2004 Training Conference, Kansas City, USA* (Mayo 2004).

[WD01] D. Wagner and D. Dean. Intrusion detection via static analysis. *In Proceedings of the IEEE Symposium on Security and Privacy. IEEE Press, Oakland, CA.* (2001).

[WDHL15] Li Wang, Chunhua Dong, Jianping Hu, and Guodong Li. Network intrusion detection using support vector machine based on particle swarm optimization. (2015).

[Web04] John Webster. Mapreduce: Simplified data processing on large clusters. *Search Storage* (2004).

[Wei99] Mark Weiser. The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.* **3**(3), 3–11 (July 1999).

[WF06] Wagner T. Y Wolstenholme P. Wagner F., Schmuki R. Modelling software with finite state machines: A practical approach. *Auerbach Publications, ISBN 0849380863* (Mayo 2006).

[WHF07] Carsten Willems, Thorsten Holz, and Felix Freiling. Toward automated dynamic malware analysis using cwsandbox. *IEEE Security and Privacy* **5**, 32–39 (2007).

[WHMH10] Gang Wang, Jinxing Hao, Jian Ma, and Lihua Huang. A new approach to intrusion detection using artificial neural networks and fuzzy clustering. *Expert Systems with Applications* **37**(9), 6225–6232 (2010).

[WL98] S.J. Stolfo W. Lee. Data mining approaches for intrusion detection. *7th USENIX Security Symposium* (1998).

[WL99] K. Mok W. Lee, S.J. Stolfo. A data mining framework for building intrusion detection models. *IEEE Symposium on Security and Privacy* (1999).

[WP99] Stephanie Forrest Warrender, Christina and Barak Pearlmutter. Detecting intrusions using system calls: Alternative data models. *IEEE Synposium on security and privacy* (1999).

[WPL+15] Jianfa Wu, Dahao Peng, Zhuping Li, Li Zhao, and Huanzhang Ling. Network intrusion detection based on a general regression neural network optimized by an improved artificial immune algorithm. *PloS one* **10**(3) (2015).

[WS02] D. Wagner and P. Soto. Mimicry attacks on host-based intrusion detection systems. *In Proceedings of the 9th ACM Conference on Computer and Communications Security. Washington DC. 255-264* (2002).

[WS14] Jeff Whitworth and Shan Suthaharan. Security problems and challenges in a machine learning-based hybrid big data processing network systems. *ACM SIGMETRICS Performance Evaluation Review* **41**(4), 82–85 (2014).

[Wu08] Kang-Feng; Yang Yi-Xian Wu, Bin; Zheng. Analysis of alert correlation in honeynet. *Journal of Beijing University of Posts and Telecommunication.* **31**(6), 63–66 (Dec. 2008).

[XN08] Dingbang Xu and Peng Ning. Correlation analysis of intrusion alerts. *Intrusion Detection Systems* **38**, 65–92 (2008).

[XNK+05] Jun Xu, Peng Ning, Chongkyung Kil, Yan Zhai, and Chris Bookholt. Automatic diagnosis and response to memory corruption vulnerabilities. In "Proceedings of the 12th ACM conference on Computer and communications security", pages 223–234. ACM (2005).

[Yad] Danny Yadron. Symantec Develops New Attack on Cyberhacking.

[yDD01] Gómez J. y Dasgupta D. Evolving fuzzy classifiers for intrusion detection. *Proceedings of the 2002 IEEE Workshop on Information Assurance, United States Military Academy, West Point, Ney York, USA* (Junio 2001).

[Yer06] A. Vasudevan; R. Yerraballi. Spike: Engineering malware analysis tools us- ing unobtrusive binary-instrumentation. *In Proceedings of the 29th Australasian Computer Science Conference* pages 311–320 (2006).

[yKR92] Porras P. y Kemmerer R. Penetration state transition analysis - a rule-based intrusion detection approach. *Eighth Annual Computer Security Applications Conference, Pag. 220-229, IEEE Computer Society Press* (Diciembre 1992).

[yKS97] Kantzavelou I. y Katsikas S. An attack detection system for secure computer systems - outline of the solution. *Proceedings of the IFIP TC11 13th International Conference on Information Security, Copenhagen, Dinamarca* (Mayo 1997).

[yLM98] Ryan J. y Lin M. Intrusion detection with neural networks. *Advances in Neural Information Processing Systems, No. 10, MIT Press, Cambridge, Massachusetts, USA* (1998).

[yLT91] Garvey T. y Lunt T. Model based intrusion detection. *Proceedings of the 14th National Computer Security Conference, Pag. 372-385* (Octubre 1991).

[yNP97] Porras P. y Neumann P. Emerald: Event monitoring enabling responses to anomalous live disturbances. *Proceddings of the 20th NIST-NCSC National Information Systems Security Conference, Pag. 353-365* (1997).

[yNP02] Russell S. y Norvig P. Artificial intelligence: A modern approach. *Second Edition, Prentice Hall, ISBN 0137903952* (Diciembre 2002).

[yNT98] Ptacek T. y Newsham T. Insertion, evasion and denial of service: Eluding intrusion detection. *Technical Report, Secure Networks Incorporated* (Enero 1998).

[YOO+13] Ting-Fang Yen, Alina Oprea, Kaan Onarlioglu, Todd Leetham, William Robertson, Ari Juels, and Engin Kirda. Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks. In "Proceedings of the 29th Annual Computer Security Applications Conference", pages 199–208. ACM (2013).

[yRG05] Giarratano J. y Riley G. Expert systems: Principles and programming. *Course Technology, Cuarta Edicion, ISBN 0534384471* (Octubre 2005).

[ySA03] Mukkamala S. y Sung A. A comparative study of techniques for intrusion detection. *Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence (ICTAI), Pag. 570* (2003).

[ySE94a] Kim G. y Spafford E. The design and implementation of tripwire: A file system integrity checker. *ACM Conference on Computer and Communications Security, Pag. 18-29* (1994).

[ySE94b] Kumar S. y Spafford E. An application of pattern matching in intrusion detection. *Technical Report 94-013, Department of Computer Science, Purdue University* (Marzo 1994).

[YSE+07] Heng Yin, Dawn Song, Manuel Egele, Christopher Kruegel, and Engin Kirda. Panorama: capturing system-wide information flow for malware detection and analysis. In "Proceedings of the 14th ACM conference on Computer and communications security", CCS '07, pages 116–127, New York, NY, USA (2007). ACM.

[ySR00] Jones A. y Sielken R. Computer system intrusion detection: A survey. *Technical Report, Computer Science Department, University of Virginia* (2000).

[yVH92] Liepins G. y Vaccaro H. Intrusion detection: Its role and validation. *Computers and Security, Vol. 11, Pag. 347-355, Elsevier Science Publishers Limited, Oxford, UK* (1992).

[yWB04] Meyer T. y Whateley B. Spambayes: Effective open-source, bayesian based, email classification system. *Proceedings of the 2004 Conference on Email and Anti-Spam (CEAS), Mountain View, California, USA* (Julio 2004).

[ZHR+07] Jingmin Zhou, Mark Heckman, Brennen Reynolds, Adam Carlson, and Matt Bishop. Modeling network intrusion detection alerts for correlation. *ACM Trans. Inf. Syst. Secur.* **10** (February 2007).

[ZL04] Amitabha Das Zhuowei Li. Visualizing and identifying intrusion context from system calls trace. *Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC'04), IEEE Computer Society* (December 2004).

[ZLK09] Chenfeng Vincent Zhou, Christopher Leckie, and Shanika Karunasekera. Decentralized multi-dimensional alert correlation for collaborative intrusion detection. *Journal of Network and Computer Applications* **32**(5), 1106–1123 (2009).

[ZPD+12] Paul Zikopoulos, Krishnan Parasuraman, Thomas Deutsch, James Giles, David Corrigan, et al.. "Harness the power of big data The IBM big data platform". McGraw Hill Professional (2012).

[ZV04] Jingyu Zhou and Giovanni Vigna. Detecting attacks that exploit application-logic errors through application-level auditing. *University of California Santa Barbara, USA* (2004).

La presente disertación ha sido concluída en Bilbao
a 29 de septiembre de 2015, siendo dedicada a la memoria de
Olga Lastra González
(01/08/1955 - † 04/06/2015)