Sistema Concurrente de Detección de Intrusiones con Correlación de Alertas en Entornos Distribuidos

SISTEMAS INFORMÁTICOS

CURSO 2012-2013

Roberto Gordo Torres
Santiago Gutiérrez Montes
Pedro Antonio Rodríguez Díaz

Director
Luis Javier García Villalba

Departamento de Ingeniería del Software e Inteligencia Artificial
Facultad de Informática
Universidad Complutense de Madrid

Madrid, Junio de 2013

Abstract

Typical scenarios of a NIDS usually are varied sized networks, from domestic to large companies. But there are also proposals to adapt it to the cloud computing. Since this kind of computing paradigm presents fairly recent security risks, we believe may be reduced with NIDS. The intrusion detection system proposed in this document proposes a series of measures to adapt a NIDS to an environment of cloud computing and motivated by two shortcomings that could present, this proposal proposes two improvements, the first of which will improve analysis speed by using parallelism at CPU and GPU and the second generate a system alert correlation. As a method for achieving these goals are assessed different ways that develop throughout this document. OpenStack will help us to deploy a cloud computing on one or more physical nodes, CUDA and OpenMP will help us to use parallelism at GPU and CPU level, and fuzzy logic will help us to label each attack. As future research lines would be the development of a sorting algorithm that exploits parallelism and optimize CPU level alert correlation.

Keywords

NIDS, cloud computing, GPU, alert correlation, OpenStack, CUDA, OpenMP, fuzzy logic.

Resumen

Los escenarios típicos de un NIDS suelen ser redes de tamaño muy variado, desde domesticas hasta de grandes empresas. Pero también hay propuestas para adaptarlos a la computación en la nube. Al ser este tipo de computación un paradigma bastante reciente presenta riesgos de seguridad que creemos que pueden ser reducidos con un NIDS. El sistema de detección de intrusos propuesto en el presente documento propone una serie de medidas para adaptar un NIDS a un entorno de computación en la nube y, motivados por dos carencias que podría presentar esta propuesta, se proponen dos mejoras, la primera de ellas será la mejora de velocidad de análisis mediante el uso de paralelismo tanto a nivel GPU como CPU y la segunda será añadirle un sistema de correlación de alertas. Como método para conseguir estos objetivos se han evaluado diferentes vías que se desarrollan a lo largo de este documento. *OpenStack* permitirá desplegar un sistema de computación en la nube sobre uno o varios nodos físicos, CUDA y OpenMP hacer uso de paralelismo a nivel de GPU y CPU, y la lógica difusa etiquetar las alertas en cada uno de los tipos de ataque. Como líneas de investigación futuras quedaría el desarrollo de un algoritmo de ordenación que explote el paralelismo a nivel de CPU y optimizar la correlación de alertas.

Palabras clave

NIDS, computación en la nube, GPU, correlación de alertas, OpenStack, CUDA, OpenMP, logica difusa.

Autorizamos a la Universidad Complutense de Madrid difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

Autor 1

Autor 2

Autor 3

Índice General

Índice de Tablas

Índice de Figuras

Capítulo 1

Introducción

El uso de internet y de las redes informáticas ha crecido en los últimos años de forma exponencial, tanto a nivel empresarial como a nivel doméstico. Sin embargo, este crecimiento del uso y las tecnologías han traído consigo un incremento aún mayor del número de ataques e intrusiones en los sistemas informáticos de todo el mundo. De ahí que la seguridad sea actualmente uno de los campos de investigación más importantes de la informática, sobre todo en el ámbito empresarial.

El Sistema Concurrente de Detección de Intrusiones con Correlación de Alertas en Entornos Distribuidos (SCDICAED) es un esfuerzo realizado por los alumnos de la Facultad de Informática de la Universidad Complutense de Madrid Roberto Gordo Torres, Santiago Gutiérrez Montes y Pedro Antonio Rodríguez Díaz como proyecto final de carrera dirigidos por el profesor Luis Javier García Villalba dentro del departamento ISIA (Departamento de Ingeniería del Software e Inteligencia Artificial) durante el curso 2012-2013, como continuación a un trabajo previo realizado dentro de las líneas de investigación del grupo.

SCDICAED es un sistema que tiene como objetivo mejorar en términos de efectividad y velocidad el conocido sistema de detección de intrusos basado en firmas Snort, así como estudiar las adaptaciones que serían convenientes para adaptarlo a entornos distribuidos. Las metas planteadas en el origen del proyecto fueron las siguientes: adaptación del NIDS a entornos distribuidos, mejorar su rendimiento mediante el uso de concurrencia tanto a nivel CPU como GPU y añadirle un sistema de correlación de alertas.

1.1. Conceptos previos

Antes de explicar cómo se ha afrontado la meta anteriormente expuesta es conveniente establecer ciertos conocimientos relativos al dominio de sistemas de detección de intrusos y del paralelismo a nivel de GPU. Se comenzará explicando en este mismo apartado conceptos necesarios tales como el de seguridad informática, amenazas, distintos componentes de lo que ha venido a llamarse mecanismos de defensa, computación distribuida, computación en la nube, NIDS, problematicas en los NIDS, GPU, el modelo de programación GPU, la computación GPU, *data mining* y lógica *fuzzy*.

1.1.1. Seguridad informática

La definición de la seguridad informática se basa en reunir en un sistema de información confidencialidad, integridad y disponibilidad [1]. La confidencialidad requiere que la información sea accesible únicamente por aquellos que estén autorizados, la integridad que la información se mantenga inalterada ante accidentes o intentos maliciosos, y la disponibilidad significa que el sistema informático se mantenga trabajando sin sufrir ninguna degradación en cuanto a accesos y provea los recursos que requieran los usuarios autorizados cuando estos los necesiten.

1.1.2. Amenazas

Un ataque es una secuencia de interacciones con el sistema que pone en riesgo cualquiera de las características arriba mencionadas. En la última década el uso de sistemas de información se ha extendido entre la población, así mismo ha aumentado la exposición de dichos sistemas ante ataques por el entorno en el que son utilizados (conexiones de acceso público, redes domesticas inadecuadamente protegidas).

Antes, los intrusos necesitaban de un conocimiento más profundo de las redes y de los sistemas informáticos para poder lanzar sus ataques pero actualmente, debido al incremento del conocimiento sobre el funcionamiento de los sistemas, los intrusos están cada vez más preparados y tienen a su disposición infinidad de herramientas con las que poder determinar las debilidades de los sistemas y explotarlas con el fin de obtener los privilegios necesarios para realizar cualquier acción dañina.

Otra fuente de riesgos proviene de los programas o aplicaciones instalados en nuestros sistemas. En muchos casos la elevada competitividad del mercado ha obligado a reducir el tiempo destinado a diseño y pruebas de producto lo que ha llevado, a pesar de las mejoras temporales que ha traído la adopción de metodologías de desarrollo, a la presencia de errores sin detectar que posibilitan la aparición de vulnerabilidades a ser explotadas por los atacantes.

1.1.3. Mecanismos de defensa

Para la correcta protección de un sistema existen diversos mecanismos tales como el cifrado y la identificación y autenticación de usuarios que permiten dotarle de integridad y confidencialidad. Por otro lado existen mecanismos que velan por la disponibilidad del sistema filtrando información, ejemplos de estos mecanismos son las listas de acceso y cortafuegos.

Como última línea de defensa de un sistema de información encontramos técnicas enfocadas a la detección de las amenazas tales como antivirus y sistemas de detección de intrusos. La característica distintiva de estas técnicas es que, al contrario de las anteriores, permiten detectar intentos de romper la seguridad de un sistema.

1.1.4. Computación distribuida

Un sistema distribuido es un tipo de sistema software cuyos componentes están desplegados en máquinas físicas separadas conectadas en red. A estas máquinas las llamaremos nodos. Dichos componentes interactúan para conseguir un propósito común de manera que la forma de obtención de este propósito es transparente al usuario final.

La computación distribuida debido a su alta escalabilidad permite obtener una potencia de cálculo superior a la que ofrecen otros sistemas de computación tales como *mainframes* o supercomputadores. Además, si llegado un punto se requiere ampliar la capacidad de cómputo del sistema, mientras que con otras soluciones no distribuidas se requeriría cambiar el equipo entero, en los sistemas de computación distribuida bastaría con añadir más nodos. También en la comparativa de tolerancia a fallos los sistemas distribuidos salen beneficiados, al no haber un único punto de fallo.

1.1.5. Computación en la nube

La computación en la nube es la evolución de varias tecnologías y paradigmas. Es un tipo particular de computación distribuida cuyas capacidades se ofrecen como servicio, con lo que el usuario final de esta tecnología no tiene que ser un experto de ninguna de las tecnologías subyacentes. Este aprovisionamiento de recursos al cliente escala con respecto a la demanda en cada momento, y se cobra por uso. En este aspecto es curiosa la comparación que se hace de la computación en la nube con el aprovisionamiento de electricidad. A comienzos de la revolución industrial, cada fábrica generaba la energía eléctrica que consumía, hasta que se empezó a comerciar con el excedente hasta que surgieron empresas únicamente dedicadas a generar electricidad. Los que más fuertemente apuestan por esta tecnología son de la opinión de que con los recursos de computación ocurrirá lo mismo que con la electricidad, lo que aun parece una apuesta arriesgada.

La computación en la nube, como la electricidad en la comparación anterior, está basada en los principios de la economía de escala, ya que los recursos ofrecidos, al ser compartidos por numerosos usuarios tienen un aprovechamiento mayor. Esto beneficia al proveedor de servicios de computación en la nube (CSP, *cloud service provider*) al aprovechar casi al 100 % sus recursos, lo cual tiene beneficios económicos y medioambientales, y al cliente, ya que solamente paga por lo que usa. Además la contratación de dichos sistemas requiere de una inversión inicial muy baja, casi nula comparada con el precio de tener que invertir en comprar y gestionar sistemas propios.

Varias teorías apuntan a que la popularidad del término computación en la nube, *cloud computing* en inglés, se debe a las divisiones de marketing de algunas compañías, que lo usaban para ofrecer alojamiento a servidores de aplicaciones, pero nosotros lo usaremos en este documento con un aspecto más amplio.

Más adelante daremos varias taxonomías basadas en los modelos de servicio, que entre otros son software como servicio, plataforma como servicio e infraestructura como servicio, y sobre sus modelos de despliegue, que son público, híbrido, privado y de comunidad, además de una definición más rigurosa proporcionada por el *National Institute of Standards and Technology* de los Estados Unidos.

1.1.6. NIDS

NIDS son las siglas de *Network Intrusion detection System*, estos IDS analizan eventos del tráfico provenientes del tráfico de red para clasificarlos adecuadamente en pertenecientes o no a un ataque. Existen diversas formas de clasificar los NIDS: por objeto de análisis, por situación de los componentes del sistema de detección de intrusismo, por tipo de análisis efectuado y por ultimo por la técnica de análisis empleada.

Un NIDS monitoriza los eventos de red, esto significa que el objeto de análisis son los paquetes y tramas de los diversos protocolos de la misma. Estos eventos de red se dividen en dos partes: cabecera y contenedor, en consecuencia se han desarrollado NIDS's que abordan el problema de la detección de intrusiones mediante el análisis de una de estas componentes dado que cada una de estas aproximaciones trae ventajas en la detección de distintos tipos de ataques.

1.1.7. Problemáticas en los NIDS

En los NIDS actualmente se dan dos carencias muy importantes:

Déficit de velocidad

Los modelos NIDS que usan unidades centrales de procesamiento (CPU) como IDS Snort están teniendo dificultades por los cuellos de botella que se crean en el sistema a raíz de que el tráfico en las redes está creciendo más rápido que la velocidad del reloj de las CPU. Estos cuellos de botella son producidos porque los NIDS no son capaces de analizar los paquetes entrantes a la red en tiempo real. Todo esto provoca que el flujo masivo de paquetes de datos sobrecargue los NIDS, llevándoles a una pérdida de paquetes en la fase de análisis y, por tanto, a su entrada en el sistema sin comprobar si son malware o intentos de intrusión, incrementando la tasa de falsos negativos.

En la última década a pesar de que se le han ido añadiendo cada vez más núcleos a las CPU no se ha conseguido subsanar el problema de los cuellos de botella puesto que el ancho de banda requerido por las infraestructuras de protección también ha ido en aumento.

Exceso de alertas

Normalmente los sistemas de detección de intrusiones emiten gran cantidad de alertas de las cuales no se tiene información. Esto provoca que el operador tenga dificultad para identificar las amenazas más peligrosas y que, en consecuencia, no pueda tomar las mejores medidas de prevención.

1.1.8. GPU

La unidad de procesamiento gráfico (GPU) es un coprocesador dedicado al procesamiento de gráficos u operaciones en coma flotante para aligerar la carga de trabajo de la CPU. De esta forma mientras gran parte del procesado de gráficos se hace en la GPU, la unidad central de procesamiento puede encargarse de otros tipos de cálculos secuenciales

[2].

1.1.9. Modelo de programación GPU

Las unidades programables de GPU siguen un modelo de programación basado en un único programa de múltiples datos. Gracias a la cantidad de núcleos de la GPU se pueden conseguir grandes mejoras de eficiencia procesando muchos elementos en paralelo con un único programa [3]. Cada elemento es independiente de los demás y, en los modelos de programación básicos, estos elementos no se pueden comunicar entre sí. En la figura 1.1 se puede observar un ejemplo del orden de ejecución de un programa concurrente en CUDA [4].

Todos los programas para una GPU deben seguir la siguiente estructura: muchos elementos en paralelo, cada uno procesado en paralelo por un único programa.

1.1.10. Computación GPU

Es el término que usamos para referirnos a la cesión del control a la GPU para que acelere cargas de cálculos que normalmente pertenecen a la CPU pero que la alta capacidad de cómputo de las GPU nos permite realizarlos a mayor velocidad, dejando para la unidad central de procesamiento aquellos cálculos que sean secuenciales.

1.1.11. *Data mining*

La minería de datos, o *Data mining*, es una metodología diseñada para generar el conocimiento a partir de datos, encontrar relaciones ocultas entre variables o prever comportamientos mediante un proceso que combina métodos y herramientas de bases de datos, estadística e inteligencia artificial aplicados sobre grandes cantidades de información.

1.1.12. Lógica *Fuzzy*

La lógica borrosa es básicamente una lógica multi-evaluada que permite valores intermedios para poder definir evaluaciones convencionales como verdadero/falso, permitiéndonos procesar datos inciertos.

1.2. Objeto de la investigación

Ya hemos explicado brevemente las características y funcionalidades de la computación distribuida, de la paralelización GPU y de la correlación de alertas. Este proyecto tendrá como meta adaptar un NIDS a las características especiales de la computación en la nube. Como herramienta base de la investigación se ha decidido usar SDARP [5], un proyecto desarrollado durante el curso 2011/2012. La razón por la que se ha elegido emplear como base a SDARP es, a parte de por ser muy preciso en la fase de análisis, por la posibilidad de poder entrenarlo contra ataques específicos. Como consecuencia de haber optado por usar como herramienta base a SDARP también se ha decidido crear otras dos vías paralelas a la meta anteriormente mencionada para suplir algunas carencias que

presenta.

Por un lado, con motivo de los problemas que están teniendo los NIDS con los cuellos de botella que se producen durante el procesamiento de paquetes entrantes a la red, y al estar presente este problema también en SDARP, se ha optado por tomar como referencia su algoritmo de análisis CPU para transformarlo en un híbrido que se sirviese del modelo de programación GPU y del paralelismo a nivel de CPU para mejorar su rendimiento. Esta mejora se basa en aumentar la velocidad de procesamiento de los paquetes de datos que entran al sistema sin recurrir a una implementación hardware.

Por otro lado, para dar mayor precisión a SDARP y motivados por el hecho de que normalmente los NIDS producen gran cantidad de alertas de las que no tenemos ningún tipo de información, se propondrá un sistema que permita correlacionar alertas, identificarlas y reducir la tasa de falsos positivos.

1.3. Estructura del trabajo

En el presente capítulo se han introducido conceptos tales como la seguridad informática, amenazas, posibles mecanismos de defensa, computación distribuida y en la nube, NIDS y sus actuales problemáticas, GPU, el modelo de programación GPU, la computación GPU, *data mining* y la logica *fuzzy*. Además se han introducido las distintas vías de este trabajo que son adaptar un NIDS a los entornos distribuidos, aumentar su rendimiento convirtiéndolo en un sistema concurrente y añadirle un sistema de correlación de alertas.

En los siguientes capítulos se tratará el estado del arte de los entornos distribuidos, de la concurrencia a nivel GPU y de la correlación de alertas (Capítulo 2). A continuación mostraremos los trabajos realizados en el proyecto SCDICAED así como los resultados obtenidos (Capítulo 3). Por último presentaremos las conclusiones y las propuestas de trabajo futuro que consideramos de gran interés para la mejora de nuestra aplicación y para la consecución de mejoras significativas en el campo de la detección de intrusos.

Adicionalmente encontramos los apéndices que contienen información relacionada con el proyecto pero explicable de manera atómica: adaptación del entorno a CUDA y adaptación *Xfuzzy*.

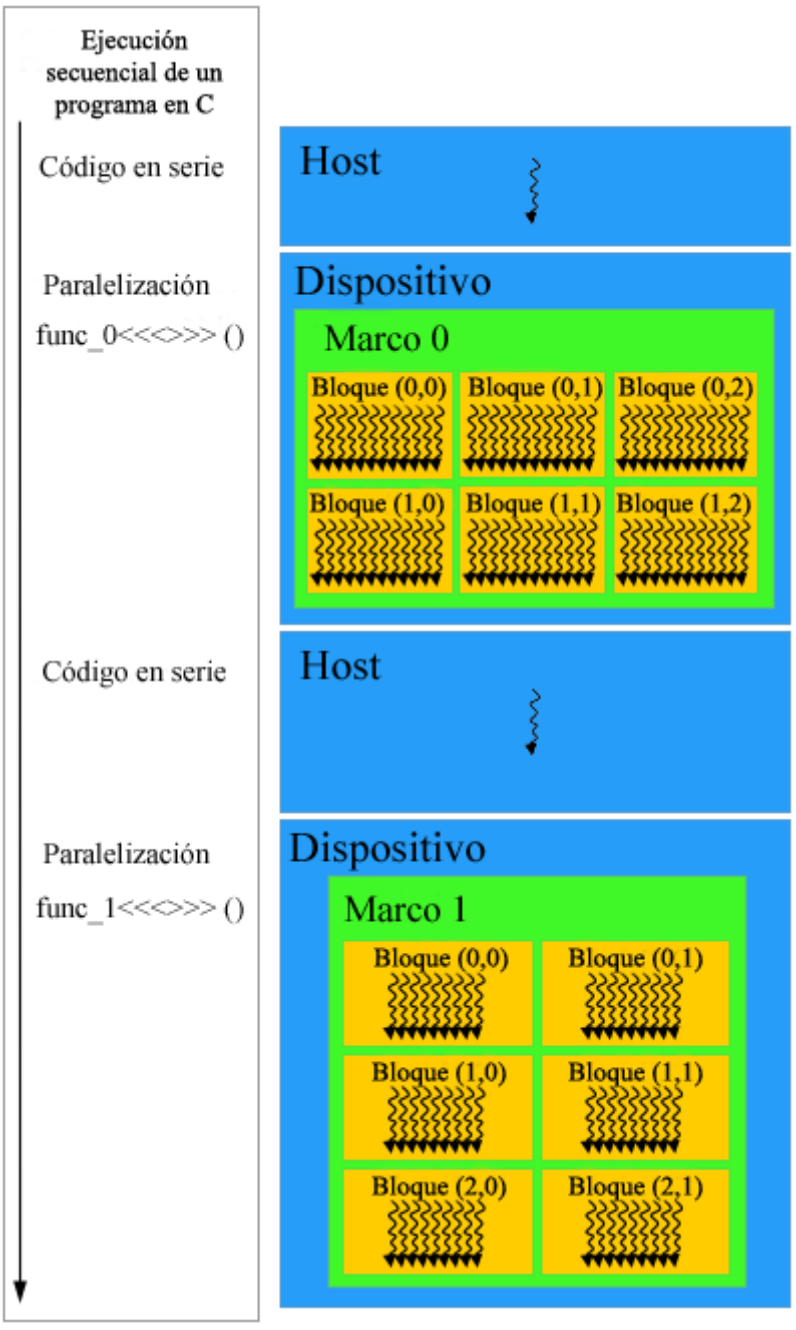

Figura 1.1: Ejemplo de estructura de ejecución de un programa en CUDA

Capítulo 2

Estado del arte

En el capítulo anterior se han definido brevemente varios conceptos sobre los que se hablará en el resto del documento. Estos conceptos sirven para dar una idea muy básica al lector. De cara a apoyar las decisiones que se tomarán en las diferentes propuestas de adaptación de un NIDS a un entorno distribuido, en la presente sección se entra en profundidad a definir el estado del arte de los distintos campos que se tratarán. Estos campos sobre los que se centrará esta sección son: sistemas de detección de intrusiones en entornos distribuidos, concurrencia y tratamiento de las alertas generadas por el NIDS.

2.1. Sistemas de detección de intrusiones en entornos distribuidos

2.1.1. Computación en la nube en profundidad. Definición y taxonomías

Como ya se comentó en la sección de conceptos básicos, en los últimos años se ha ido desarrollando muy rápidamente una nueva manera de ofrecer recursos de computación a diferentes niveles llamada computación en la nube *(Cloud Computing)*. Este nuevo paradigma de computación distribuida, aun siendo actualmente ofrecido por numerosísimas empresas y organizaciones, está aún en evolución. Por esto se usará la definición del NIST (*National Institute of Standards and Technology*, de EEUU) [6] para definirlo, enumerar aspectos importantes, y establecer una taxonomía simple.

Según el NIST (*National Institute of Standards and Technology*, de EEUU):

> "La computación en la nube es un modelo para proveer acceso ubicuo, conveniente y bajo demanda a un conjunto compartido de recursos de computación a través de la red (por ejemplo: redes, servidores, almacenamiento, aplicaciones y servicios) que pueden ser rápidamente asignados y liberados con un esfuerzo mínimo de gestión o interacción con el proveedor. Este modelo está compuesto por cinco características esenciales, tres modelos de servicio y cuatro modelos de despliegue."

Características esenciales:

Aprovisionamiento bajo demanda y auto servicio: Un cliente puede obtener unilateralmente recursos de computación incluso sin requerir de interacción humana entre este y el proveedor.

Amplio acceso a la red: Estas capacidades son ofrecidas sobre la red, y accedidas a través de mecanismos estandarizados, promoviendo su uso por plataformas cliente muy heterogéneas.

Agrupamiento de recursos: Los recursos de computación del proveedor son agrupados para servir a múltiples usuarios, asignándose y reasignándose los diferentes recursos físicos y virtuales dinámicamente según las necesidades de los clientes. Hay una independencia de la localización de los recursos físicos, por la cual el cliente generalmente no tiene control o conocimiento sobre dónde estarán situados dichos recursos, aunque sí se podría especificar la localización a un alto nivel de abstracción (país, estado o centro de datos). Ejemplos de estos recursos son almacenamiento, memoria, procesamiento y ancho de banda de red.

Elasticidad: Las capacidades se pueden asignar y liberar elásticamente, en algunos casos automáticamente para escalar con la demanda. Para el cliente, los recursos disponibles parecen ser ilimitados.

Servicio de medida: Los sistemas de computación en la nube automáticamente controlan y optimizan el uso de recursos usando medidas a un nivel de abstracción apropiado para el tipo de servicio ofrecido (almacenamiento, procesamiento, ancho de banda, cuentas de usuario activas...). El uso de recursos puede ser medido y controlado dando transparencia acerca del servicio utilizado tanto al cliente como al proveedor. Típicamente la función más importante de este servicio de medida es calcular el precio a cargar al cliente por el uso.

Modelos de Servicio:

Software como servicio (*Software as a service*, SaaS). La capacidad ofrecida al cliente es usar las aplicaciones del proveedor, que son ejecutadas sobre una infraestructura de computación en la nube. Las aplicaciones son accesibles desde múltiples clientes, tanto ligeros (como un navegador en el caso por ejemplo de acceder al correo vía web) como pesados como en forma de interfaces ofrecidas a otros programas. En este modelo el cliente no gestiona ni controla la infraestructura de computación en la nube subyacente, como la red, los servidores, los sistemas operativos, el almacenamiento. En todo caso podrá acceder a algunos niveles de configuración de la aplicación que esté usando.

Plataforma como servicio (*Platform as a service*, PaaS). La capacidad ofrecida por este modelo es desplegar en la infraestructura de computación en la nube aplicaciones (creadas por el cliente o compradas) que hayan sido programadas usando lenguajes, librerías, servicios y herramientas soportadas por el proveedor. El cliente no gestiona ni controla la infraestructura de computación en la nube subyacente incluyendo redes, servidores, sistemas operativos, o almacenamiento; pero tiene el control sobre las aplicaciones desplegadas y probablemente sobre parámetros de configuración del entorno.

Infraestructura como servicio (*Infraestructure as a service*, IaaS). La capacidad que se provee al consumidor es procesamiento, almacenamiento, redes, y otros recursos de computación fundamentales donde el cliente puede desplegar y ejecutar el software que desee, lo que puede incluir sistemas operativos y aplicaciones. El consumidor no gestiona ni controla la infraestructura de computación en la nube subyacente, pero tiene control sobre sistemas operativos, almacenamiento y aplicaciones desplegadas. Probablemente también tenga control sobre algún componente de red como el cortafuegos.

Modelos de despliegue:

Privado. La infraestructura de computación en la nube está ofrecida para uso exclusivo de una sola organización, que puede tener varios clientes (típicamente departamentos dentro de la misma empresa). Puede ser propiedad de la propia organización, de una externa o una combinación de ambas. Puede estar situada dentro o fuera de las instalaciones.

Comunidad. La infraestructura de computación en la nube está ofrecida para el uso exclusivo de una comunidad de usuarios de organizaciones que comparten intereses (tareas, requisitos de seguridad, política, ...). Puede ser propiedad de una o varias de las organizaciones de la comunidad, de una organización externa o una combinación de ambas. Puede estar situada dentro o fuera de las instalaciones.

Público. La infraestructura de computación en la nube es ofrecida para el uso por el público general. Puede ser propiedad de una organización académica, gubernamental o lucrativa, o combinación de alguna de ellas. Está situada en las instalaciones del proveedor de servicios.

Híbrido. La infraestructura de computación en la nube está formada por una composición de dos o más infraestructuras de computación en la nube diferentes (privadas, públicas o de comunidad) que siguen siendo entidades diferenciadas pero están unidas por tecnología (libre o propietaria) que permite la portabilidad de los datos y/o las aplicaciones entre ellas (como el balanceo de carga entre infraestructuras de computación en la nube cuando alguna de ellas está cerca de su límite de recursos).

2.1.2. Crecimiento de la computación en la nube

La computación en la nube atrae a clientes dada su gran elasticidad y escalabilidad, además de otros beneficios que se extraen de la definición de computación en la nube del NIST, pagando por ello únicamente lo que se usa a precios relativamente bajos. Comparado con la creación de un centro de proceso de datos propio o la ampliación del existente, los clientes pueden ahorrar bastante en términos de dinero y mano de obra migrando parte de sus servicios informáticos tales como almacenamiento o proceso de datos a la nube.

Considerando los beneficios que la computación en la nube puede aportar a la empresa, no es de extrañar el crecimiento en su adopción, hasta ahora y en un futuro. La consultora Gartner, como publica Forbes [7], cifra la tasa de crecimiento anual compuesto del gasto en servicios de computación en la nube pública en un 18.6 % en 2012, suponiendo 110.300 millones de dólares dicho gasto. En concreto dicho informe estima que hasta 2016 el crecimiento en el gasto en infraestructura como servicio (IaaS) tendrá un crecimiento anual compuesto hasta 2016 del 41,3 %, siendo el área de computación en la nube con mayor crecimiento según Gartner.

Este crecimiento en la demanda en estos servicios trae consigo un crecimiento en la necesidad de protección de este tipo de infraestructuras, siendo el crecimiento previsto de servicios de seguridad en la nube de un 26.7 % medio anual hasta 2016.

2.1.3. Riesgos de seguridad

Aunque la adopción de estos servicios supone muchas ventajas, entre ellas un ahorro en términos económicos y de recursos humanos, trae consigo riesgos de seguridad inherentes a estos entornos distribuidos. Para enumerar estos riesgos, usaremos la lista del top 9 de amenazas de la *Cloud Security Alliance* en 2013 [8] explicando cada una de ellas.

El cambio de pensar en servidores a pensar en servicios está cambiando la manera en la que los departamentos de tecnología piensan, diseñan y despliegan sus aplicaciones. Y aunque se están haciendo esfuerzos para intentar clasificar y comprender estos nuevos riesgos, de algunos de ellos no se ha visto su impacto aún.

Aunque las amenazas a la seguridad son muy numerosas, este documento se centra en aquellas relacionadas con la naturaleza compartida, distribuida y bajo demanda de la computación en la nube. Nos extenderemos más en aquellas amenazas contra las cuales irá enfocada la propuesta.

Robo de datos

La computación en la nube añade nuevas formas de robo de datos debido a la naturaleza distribuida y multiusuario (en varios modelos de despliegue, sobre todo en el púbico) del sistema.

Pérdida de datos

Este riesgo, aparte de por desastres físicos o borrados accidentales de los datos, puede ser producido por un atacante malicioso. Un remedio para evitar el robo de datos sería cifrarlos, pero así, con la posibilidad de perder las claves se crearía un nuevo riesgo de pérdida de datos.

Robo de cuentas

Este no es un riesgo que introduzca la computación en la nube pero hay que tenerlo muy en cuenta. De hecho, ha pasado de estar en el número 6 de la lista de riesgos de la *Cloud Security Alliance* publicada en el año 2010, a estar en el puesto 3 de la presente lista, del año 2013. El *phishing*, la explotación de vulnerabilidades software, etc, siguen dando resultado a los atacantes. Sirvan como ejemplo los casos en los que Amazon, en 2009 y 2010 fue víctima de distintos ataques, entre ellos XSS (*cross site scripting*) [9], lo que permitió robar bastantes cuentas de usuario. También a causa de riesgos de este tipo, se encontró en los servidores de Amazon una vulnerabilidad que permitió ejecutar varios nodos de procesamiento de la *botnet Zeus* en 2009 [10].

Con las credenciales robadas se permite el acceso a áreas críticas de los servicios desplegados, permitiendo poner en peligro su confidencialidad, integridad y disponibilidad.

APIs inseguras

Los proveedores de servicios de computación en la nube (CSP) ofrecen a los usuarios interfaces o API's para gestionar los servicios de la nube. Estas interfaces deben estar diseñadas para soportar intentos de violar las políticas de seguridad tanto intencionada como no intencionadamente. Confiar en API's débilmente protegidas crea riesgos de todo tipo. Accesibilidad, disponibilidad, confidencialidad y la medida del uso de los servicios en la nube se pueden ver alteradas por vulnerabilidades en este campo.

Denegación de servicio

Los ataques de denegación de servicios (DOS, DDOS en caso de ser distribuidos) privan a usuarios legítimos del sistema de acceder a recursos debido a su agotamiento. En estos casos el atacante provoca una ralentización de los sistemas. La novedad en los entornos de computación en la nube, es que debido a que los CSP's cobran por acceso a disco y por capacidad de procesamiento, aunque un ataque de denegación de servicios puede no bloquear el acceso por las altas capacidades de ancho de banda de los CSP's, sí que se puede incrementar visiblemente la facturación al cliente por dicho aumento de uso.

Ataques desde dentro del sistema

Son los causados por un empleado malicioso. Según el CERT (el servicio de respuesta ante emergencias informáticas de la universidad Carnegie Mellon, Pittsburgh, EEUU), una amenaza interna o empleado malicioso (*malicious insider* en inglés) es un empleado o ex empleado de la compañía, contratante o compañero de negocio, que tiene o ha tenido acceso a los datos, sistemas o redes de la organización, e intencionadamente ha hecho mal uso de este acceso de manera que se pone en peligro la confidencialidad, accesibilidad o disponibilidad de los datos o sistemas de la organización [11]. Dicho esto es fácil ver el riesgo que supone un empleado malicioso y un sistema en la nube mal diseñado. Los sistemas cuya seguridad depende únicamente del CSP son un grave riesgo aquí, siendo muy recomendable en esos casos algún tipo de auditoría.

Abuso de servicios Cloud

Es enorme la cantidad de recursos que la computación en la nube puede poner a disposición de empresas de todos los tamaños, que de otra manera no tendrían acceso a esta capacidad. De todas formas, no siempre todo este poder se usa con buenos propósitos. En este punto se plantea el riesgo del uso de la nube para reventar claves, distribuir *malware* o llevar a cabo ataques DDOS, como en el ejemplo citado anteriormente sobre la *botnet Zeus*. El problema aquí está en ver cómo se puede controlar el mal uso de estas tecnologías.

Mala evaluación de la propuesta

Muchas empresas se han lanzado o se van a lanzar a mover sus sistemas a la nube por sus promesas de disponibilidad, adaptabilidad y eficiencia en términos económicos. Pero demasiadas de estas empresas no comprenden enteramente esta tarea.

Vulnerabilidades ligadas a la compartición de recursos

La escalabilidad y rentabilidad económica de la computación en la nube va intrínsecamente ligada a la compartición de recursos. En la mayoría de los casos, aunque no siempre, dicha compartición se ofrece mediante virtualización. La virtualización es un método para crear una versión virtual a partir de otra real de algún recurso de computación, como almacenamiento, capacidad de enrutamiento e incluso una computadora entera. Las componentes que son compartidas (cachés de la CPU, GPU, RAM...) no fueron diseñadas para ofrecer un gran aislamiento para arquitecturas multiusuario, como en el caso de IaaS, ni plataformas re-desplegables, como en el caso de PaaS, ni aplicaciones multiusuario en el caso de SaaS, por lo que esta amenaza de seguridad es común a todos los modelos de despliegue.

Desde la *Cloud Security Alliance* se recomienda una estrategia de defensa en profundidad que incluya protección de los servicios de computación, de red, de almacenamiento, gestión de identidad y monitorización, sea cual sea el modelo de despliegue. Una ruptura en la seguridad de alguno de los componentes podría suponer la puerta de entrada a un ataque mayor. De ahí viene la importancia de este riesgo de seguridad, porque potencialmente podría afectar al sistema entero.

2.1.4. Orquestadores de computación en la nube

En el camino por conocer qué decisiones tomar para adaptar un NIDS a un entorno distribuido, se debe estudiar los distintos sistemas software que hay en la actualidad para ofrecer las funcionalidades requeridas de un sistema de computación en la nube. Hay que recordar que los NIDS trabajarán detectando intrusiones a nivel de infraestructura, con lo que se debe buscar de entre la oferta de sistemas que ofrecen la posibilidad de poder crearse una infraestructura de computación en la nube propia (de comunidad, híbrida o privada), es decir, los que ofrecen infraestructura como servicio (IaaS). Estos sistemas son llamados orquestadores de computación en la nube, y hay múltiples alternativas en el mercado, tanto privativas como de código abierto. De entre los privativos, el orquestador más popular es la solución de la empresa VMWare, llamada vCloud Director. Bastante menos extendidas están las soluciones que ofrecen IBM con SmartCloud, Microsoft con SystemCenter o Citrix con CloudStack/ CloudPlatform. En cuanto a los orquestadores de código abierto, menos extendidos pero con un crecimiento muy rápido en los últimos meses podemos destacar varios, como *Eucalyptus, OpenNebula, CloudStack, Nimbus y Openstack.*

Estos sistemas *software* distribuidos constan de varios módulos cada uno de los cuales ofrece una funcionalidad. Será necesario saber qué partes se encargan de qué funcionalidades de cara a saber qué riesgos presenta cada una de las partes y su nivel de exposición a amenazas. Típicamente, los sistemas orquestadores tienen al menos módulos que ofrecen estos servicios:

Controlador de computación en la nube: Es el encargado de gestionar y automatizar los recursos de computación usando para ofrecerlos virtualización (mediante distintos hipervisores como KVM, XenServer y muchos otros) o sin virtualización. Un hipervisor o monitor de máquina virtual es un sistema que crea y ejecuta máquinas virtuales.

Almacenamiento de objetos y/o bloques: Gestiona el almacenamiento secundario a

nivel de bloque y/o de objetos.

Gestión de red: Este servicio se encarga de gestionar las redes virtuales que se necesiten y las direcciones IP

Identificación: Se ocupa de la gestión de identidades junto con los servicios a los que se tiene acceso.

Gestión de imágenes: Gestiona las imágenes que se ofrecen para iniciar máquinas virtuales, así como se ocupa de guardar copias de seguridad de otras imágenes cuando sea necesario.

Interfaz: Típicamente se suele incluir una interfaz de comunicación visual con la infraestructura, aunque la mayoría de las interacciones con ésta se harán directamente con las APIs ofrecidas.

Medida: se encarga de llevar la cuenta de los servicios usados por cada usuario de la infraestructura para luego poder llevar a cabo la facturación.

2.1.5. Trabajos relacionados

En esta sección veremos las distintas propuestas de adaptación disponibles para añadir la protección de un NIDS a un entorno distribuido, en concreto un entorno de computación en la nube.

- K. Vieira et al. (2010)[12]: Este estudio sobre los IDS para *grid* (otro tipo de sistema de computación distribuida) y computación en la nube propone que cada nodo de la infraestructura distribuida detecte los eventos locales que pudiesen constituir una violación de la seguridad y alerte a los demás. Cada nodo almacenará una copia de una base de firmas y de comportamientos, que será usada por el auditor del sistema (también local a cada nodo) para tomar decisiones sobre el tráfico que controla. Las alertas son comunicadas a los demás nodos para actualizar sus bases de conocimiento. El esquema se puede ver en la Figura 2.1:
- P. K. Shelke et al. (2012) [13]: Este documento señala la necesidad de que el NIDS sea multihilo de cara a mejorar su velocidad. Si el NIDS no fuera suficientemente rápido, la red se ralentizaría, o se podrían dejar paquetes sin analizar, lo que constituye una vulnerabilidad. Además indica las ventajas de que exista una tercera empresa de auditoría entre el proveedor de servicios de computación en la nube (CSP) y el cliente, basándose en que, como el que gestiona la seguridad de la infraestructura no es el cliente, sino el CSP, éste podría ocultar los ataques sufridos para preservar su imagen. Se propone un NIDS basado en firmas y detección de anomalías, con estas dos particularidades citadas anteriormente: concurrencia en el proceso de paquetes y gestión por parte de una tercera empresa.
- S. Roschke et al. (2009) [14]: Este documento propone un NIDS distribuido de manera parecida con comunicación entre sus partes, dando varios ejemplos de los posibles ataques, como *cross site scriting* (XSS) y *buffer overflow* y sus consecuencias sobre el sistema. También habla de la importancia de correlacionar las alertas para detectar más fácilmente ataques a gran escala desde/hacia un nodo concreto, como DoS

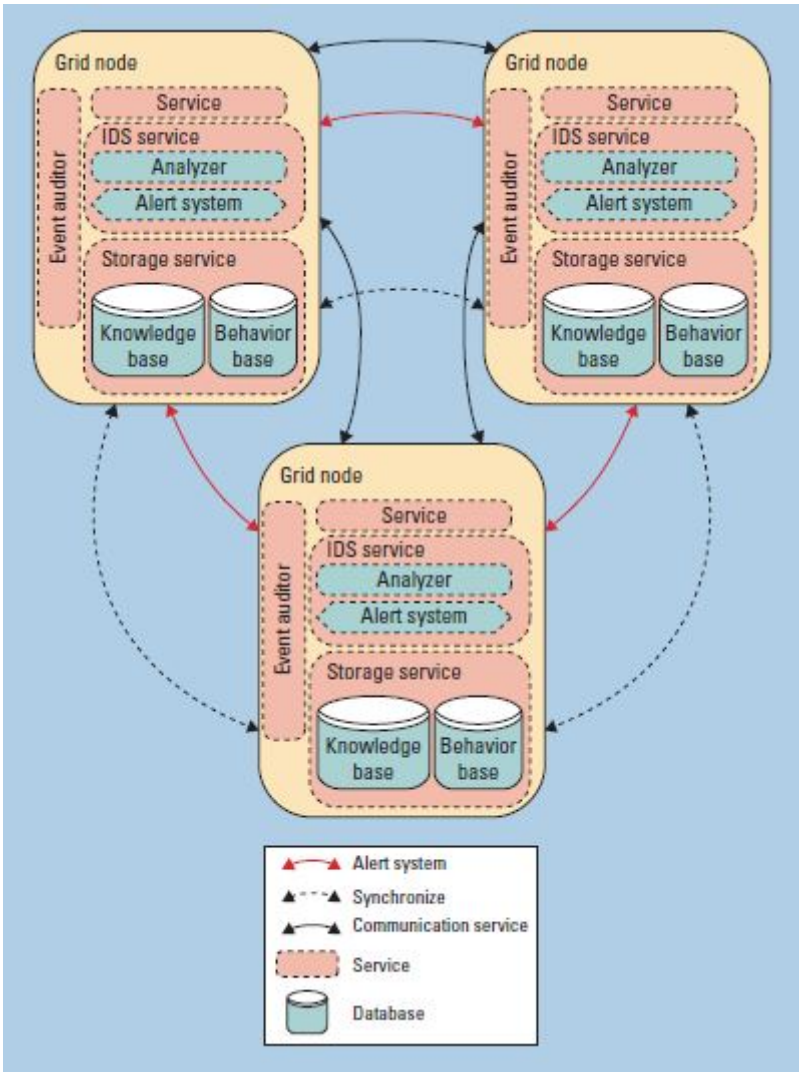

Figura 2.1: Esquema de propuesta para NIDS en *Grid* y en sistemas de computación en la nube

o DDoS [15]. Como característica esencial de la adaptación señala dotar al NIDS de escalabilidad, eficiencia y adaptarlo al contexto basado en virtualización. De los distintos trabajos relacionados se elige en esta propuesta con la necesidad de concurrencia en la ejecución del NIDS debido a los mayores volúmenes de datos que soporta, la necesidad de correlacionar las alertas, y adaptar los NIDS a las amenazas especiales a las que están expuestas las tecnologías de computación en la nube. Además también será importante elegir los emplazamientos que controlará el NIDS.

2.2. Concurrencia

La definición de computación concurrente está basada en la ejecución de forma simultánea de múltiples tareas. El desarrollo de aplicaciones dentro del ámbito científico que aprovechan la concurrencia a nivel CPU surgió hace bastante tiempo pero a nivel

GPU es bastante reciente.

La función original de las GPUs fue el procesamiento de gráficos 3D, ya que en distintos campos, como el de los videojuegos, existía cierta demanda para poder generar imágenes suaves y vivas en tiempo real. Esta mejora en el tratamiento de gráficos se consiguió añadiendo un mayor número de núcleos de procesamiento a los procesadores, ya que de esta forma se podían realizar cálculos de forma paralela, aumentando drásticamente su potencia de cálculo [16]. Esta vía de mejora, actualmente, está siendo la más seguida por los desarrolladores, pasando a un segundo plano la de aumentar el rendimiento de un único hilo, y en el futuro está previsto que este tendencia continúe [2].

El desarrollo de aplicaciones que aprovechan la capacidad de computo de las GPUs con fines no gráficos es bastante reciente y recibe el nombre de GPGPU (*General-Purpose computation on Graphics Processing Units*)[17]. Este tipo de investigaciones provienen sobre todo del ámbito científico y de la simulación [16].

2.2.1. Fabricantes de GPU

Los dos más destacados son AMD [18], compañía estadounidense que en el año 2006 compró ATI Technologies, y NVIDIA [19], empresa multinacional especializada en el desarrollo de unidades de procesamiento gráfico [3]. En la figura 2.7 se puede observar el avance de estas dos compañías en sus tarjetas graficas.

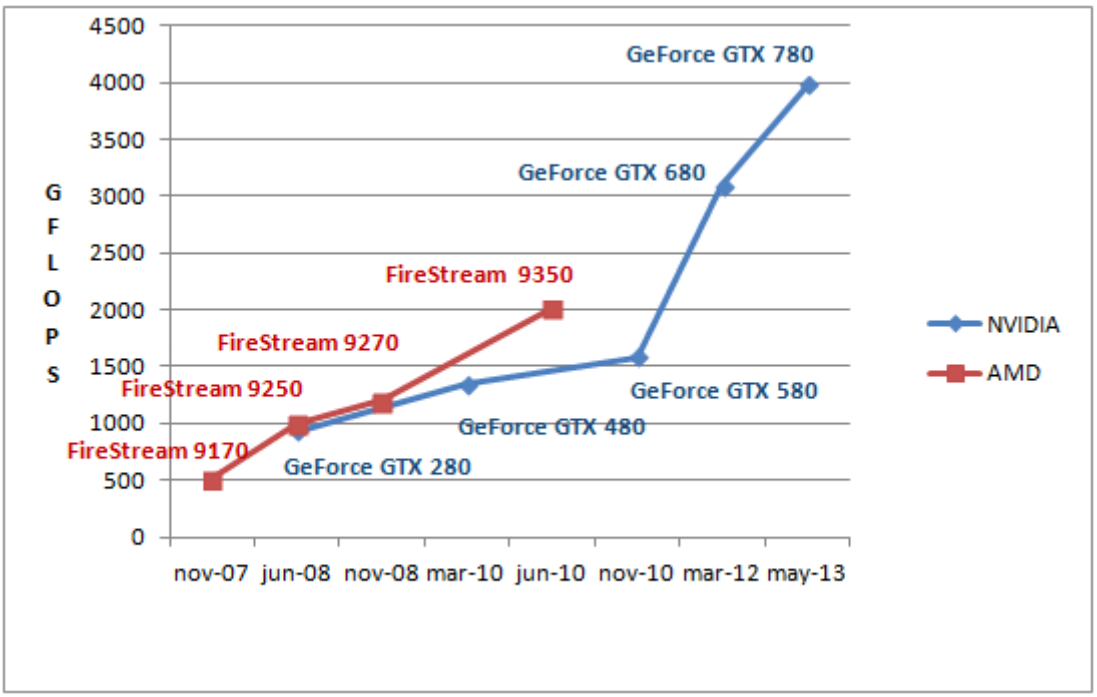

Figura 2.2: Comparativa entre las tarjetas graficas AMD y NVIDIA

AMD

En noviembre de 2006, AMD anuncio la primera generación de la serie *FireStream*. Se trataba de un procesador basado en *PCI Express* con 1 GB de memoria GDDR3 para computación con características de procesamiento de alto rendimiento optimizado.

En 2007, AMD presentó su solución de procesamiento de segunda generación, la AMD *FireStream 9170* [20]. *FireStream 9170* revolucionó la industria de GPGPU al ser la primera solución de procesamiento que ofreció compatibilidad de puntos flotantes en hardware.

En junio de 2008, AMD presentó el primer producto de su línea de soluciones de procesamiento de tercera generación, la AMD *FireStream 9250* [21]. *FireStream 9250* fue la primera solución de procesamiento que rompió la barrera de 1 TFLPS. En noviembre de ese mismo año AMD presento *FireStream 9270* [22], que superó a su versión anterior alcanzando los 1,2 TFLOPS.
Finalmente en Junio de 2010 AMD presentó *FireStream 9350* [23], la que supondría la última de la serie *FireStream* y que alcanzó 2,016 TFLOPS.

NVIDIA

En noviembre de 2006 NVIDIA presentó al mercado la GPU *GeForce 8800 GTX* [24], esta nueva arquitectura vino acompañada de la presentación de CUDA [4].

En junio de 2008 presentaron *GeForce GTX 280* [25], que supuso un gran avance para la compañía ya que alcanzó los 933 GFLOPS. Se trataba de un procesador basado en *PCI Express* con 1 GB de memoria GDDR3 para computación con características de procesamiento de alto rendimiento optimizado.

En marzo de 2010 presentaron *GeForce GTX 480* [26] con 1028MB de memoria GDDR5 y que alcanzo los 1,34TFLOPS.

En Mayo de 2013 dieron a conocer *GeForce GTX 780* [27], actualmente es la última de las series *GeForce GTX*, cuenta con una capacidad de 3,9TFLOPS y 3GB de memoria GDDR5.

2.2.2. APIs de programación GPU

En los últimos años se han desarrollado diversas APIs de programación para computación GPU, por un lado los fabricantes de GPUs han sacado soluciones específicas para su arquitectura mientras que por otro lado nos encontramos con iniciativas como la del grupo Khronos [28] para crear un lenguaje multiplataforma para el desarrollo de aplicaciones que hagan uso de la GPU.

OpenCL

OpenCL está compuesto por una interfaz de programación de aplicaciones y un lenguaje de programación que, unidos, permiten desarrollar aplicaciones con paralelismo a nivel de datos y de tareas que pueden ejecutarse tanto en la CPU como en la GPU. El lenguaje

está basado en C99, extendiendo su funcionalidad en algunos aspectos y limitándola en otros.

Fue desarrollado inicialmente por Apple Inc. y refinado en colaboración con los equipos técnicos de AMD, IBM, Intel y Nvidia. Apple presentó esta propuesta inicial al grupo Khronos y el 16 de Junio de 2008 se creó el *Khronos Compute Working Group* [29] con representantes de CPU, GPU, procesadores integrados y compañías de *software*. Este grupo trabajó durante cinco meses para terminar los detalles técnicos de la especificación de OpenCL 1.0 (18 de Noviembre de 2008). Esta especificación técnica fue revisada por los miembros de Khronos y aprobada para su publicación el 8 de Diciembre de 2008 [30]. Actualmente la última especificación técnica se encuentra en la revisión 19 y fue publicada el 14 de Noviembre de 2012 [31].

CUDA

En Noviembre de 2006 NVIDIA introdujo CUDA, una plataforma de uso general de cálculo paralelo y modelo de programación que aprovecha el motor de cálculo paralelo NVIDIA GPU para resolver problemas complejos de una forma más eficiente que una CPU.

El modelo de programación paralela en CUDA está diseñado para que suponga una curva de aprendizaje baja para los programadores familiarizados con los lenguajes de programación estándar como C. Las diferencias entre el lenguaje de programación usado en CUDA y C radican en que este ha sido usado como base pero ha sido extendido con algunas funcionalidades, como pueden ser jerarquía de grupos de hilos y de memoria, y el hacer uso de sincronización a nivel de bloques de hilos.

2.2.3. APIs de programación CPU

OpenMP [32] es una API que nos permite añadir concurrencia a las aplicaciones mediante paralelismo con memoria compartida. Se basa en la creación de hilos de ejecución paralelos compartiendo las variables del proceso padre que los crea.

2.2.4. Trabajos relacionados

Ha habido diversas investigaciones que han intentado adaptar algoritmos de cadenas coincidentes a las unidades de procesamiento gráfico (GPU):

- Huang et al. (2008) [16] En este trabajo se propone un algoritmo de coincidencias de patrón múltiple para los sistemas de detección de intrusiones de red basado en la GPU. A nivel de paralelismo se hace uso de la alta capacidad de cómputo de la GPU para inspeccionar el contenido del paquete en paralelo.

- Wu et al. (2009) [33] En este trabajo se propone la idea de utilizar un filtrado previo para reducir la carga de trabajo de los NIDS. El objetivo es el de diseñar un método de pre filtrado que haga uso del paralelismo de la GPU.

- Lin et al. (2010) [34] En este trabajo proponen conseguir una mejora de la velocidad mediante el uso de un nuevo algoritmo de búsquedas de cadenas que se beneficia del paralelismo de la GPU.

Otros trabajos relacionados que combinan los NIDS y CUDA son:

- Visialidis et al. (2008) [35] En este artículo se presenta un sistema de detección de intrusiones basado en el NIDS de código abierto Snort que explota el poder computacional de las tarjetas gráficas modernas para realizar en él la búsqueda de patrones, que es la parte más costosa para la CPU, y así incrementar el rendimiento general de procesamiento. Su prototipo, llamado Gnort, logra un rendimiento máximo de 2,3 Gbps utilizando trazas de red sintéticos, mientras que cuando las pruebas se hicieron con trafico real supero a Snort por un factor de dos. Los resultados obtenidos en este artículo sugieren que las tarjetas gráficas modernas pueden ser utilizados eficazmente para acelerar los sistemas de detección de intrusos, así como otros sistemas que involucran operaciones de coincidencia de patrones.
- Visialidis et al. (2009) [36] En este trabajo se propone un motor de búsqueda de expresiones regulares para unidades de procesamiento gráfico (GPU), ya que las características de estas permiten una búsqueda eficiente de múltiples entradas de forma simultánea comparándolas contra un gran conjunto de expresiones regulares.
- Visialidis et al. (2011) [37] En este artículo se presenta una arquitectura de multiparalelismo de detección de intrusiones hecha para redes de alta velocidad. Para hacer frente a las mayores necesidades de rendimiento de procesamiento, este sistema paraleliza el procesamiento y análisis de tráfico de red mediante tres niveles: el uso de tarjetas de red de múltiples colas, el uso de múltiples CPUs y el uso de múltiples GPUs. Este diseño evita el bloqueo, optimiza la transferencia de datos entre las diferentes unidades de procesamiento, y acelera el procesamiento de datos mediante la asignación de diferentes operaciones a las unidades de procesamiento en función de sus características.
- Nordhaug (2012) [38] La meta de este trabajo fue la de aprovechar la capacidad de computo de las GPUs para acelerar sistemas de detección de intrusismo en redes como Snort, y para ello se sirvieron de la tecnología CUDA.

Hemos decidido emplear en el desarrollo CUDA ya que en función de los resultados de los trabajos anteriormente expuestos y de que CUDA, al ser una API propia del fabricante y, por lo tanto, haber recibido mucho más apoyo para su correcto funcionamiento de forma optimizada que las de código libre, nos permitirá obtener mejores rendimientos en el apartado de concurrencia a nivel GPU. Por otro lado mantener el uso de OpenMP para mejorar el rendimiento a nivel CPU continúa siendo la mejor opción.

2.3. Minería de datos en IDS y Correlación de alertas

Actualmente los IDS producen una vasta información sobre las anomalías que detecta. Para poder estudiar estos datos es necesario clasificarlos y transformarlos en datos que puedan ser usados posteriormente. El extenso desarrollo de la minería de datos que dispone de una amplia variedad de algoritmos, extraído de los campos de la estadística, reconocimiento de patrones, aprendizaje automático, y base de datos ha motivado su uso

en la investigación.

En el campo de la seguridad se ha usado extensamente técnica de minería de datos para todo tipo de usos a lo largo de la historia. A continuación se describirá las distintas técnicas de minería de datos usadas en los sistemas de detección de anomalías.

Dependiendo de las funciones de los algoritmos de minería de datos se puede resumir en estos modelos [39] :

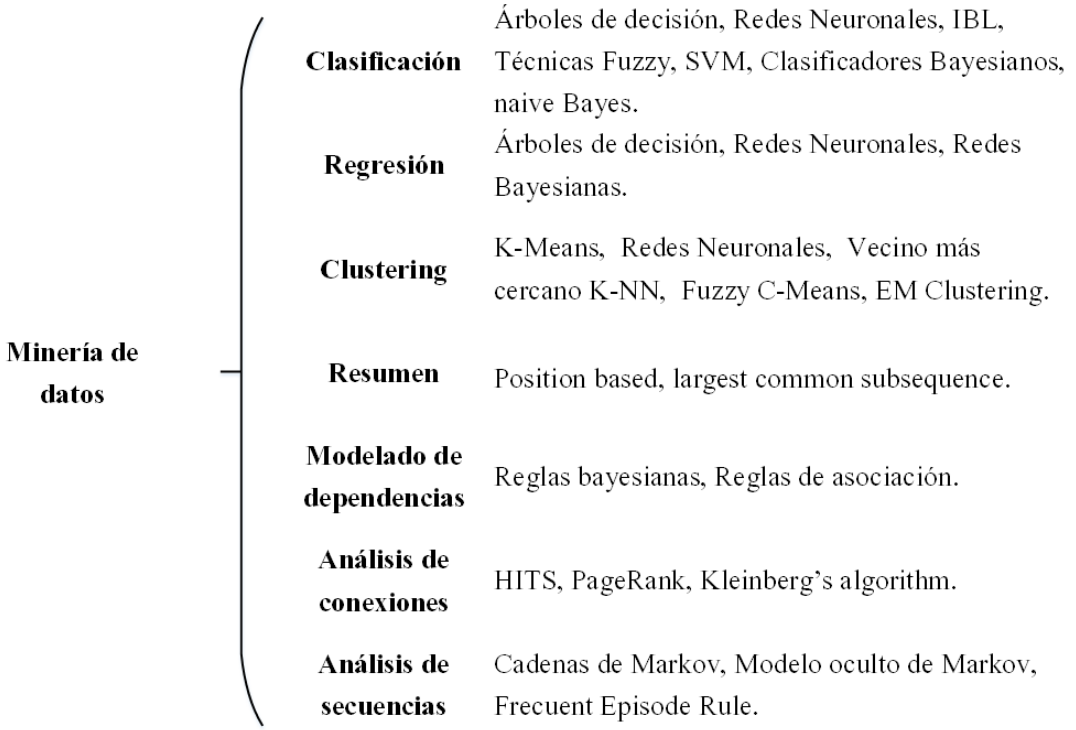

Figura 2.3: Clasificación de técnicas de minería de datos

- Clasificación: Etiqueta un caso entre varias clases o categorías predefinidas. Los sistemas de clasificación se generan a través de un amplio abanico de algoritmos. Se puede dividir en tres tipos de algoritmo: Por un lado tenemos extensiones de discriminación lineal (como perceptrón multicapa, discriminación lógica), otro tipo son árboles de decisión y métodos basados en reglas (como C4.5, AQ, CART), y el último son los estimadores de densidad (Naïve Bayes, k-nearest neighbor, LVQ).
- Regresión: Donde se busca un modelo el cual pueda predecir un conjunto de valores para una determinada variable.
- Clustering: Trata de clasificar los datos en grupos o clases usando los rasgos de los datos con unos patrones comunes. Se busca agrupaciones de los datos usando medidas de similitud, densidad de probabilidad o distancia.
- Resumen: Determina de la información que dispone cada una de las subdivisiones del conjunto de datos de entrada.

- Modelado de dependencias: relaciona las dependencias más relevantes entre las variables de una base de datos, para ello se pueden usar ciertas dependencias especificas de una variable local, una variable cuantitativa.

- Análisis de conexiones: determina las relaciones o vínculos entre campos de la base de datos. Cuya finalidad es el de encontrar las correlaciones entre campos satisfaciendo el umbral de confianza.

- Análisis de secuencias: El objetivo es modelar los estados del proceso generando la secuencia, o extraer y describir desviaciones y tendencias sobre el tiempo.

2.3.1. Minería de datos en IDS

A lo largo de la historia, desde la creación de los IDS se han usado numerosas técnicas de minería de datos desde Wenkee Lee y Sal Stolfo [40] en 1998 hasta ahora. Normalmente estos modelos se dividen principalmente en algoritmos de clasificación (aprendizaje supervisado) y *clustering* (no supervisado), aunque también se han encontrado técnicas diferentes que mencionaremos más adelante.

Uno de los rasgos más transcendentales en la minería de datos es el paradigma de aprendizaje de los sistemas. Por una parte tenemos el aprendizaje supervisado, en el cual se utilizan estimaciones basadas en un conjunto previo de entrenamiento, del cual se sabe la clase a la que pertenece. Esta estimación se obtiene a través del uso de varios algoritmos sobre un conjunto de registros de ataque. Por otra parte el aprendizaje no supervisado, también conocido como *clustering* o agrupación. Para el diseño de estos sistemas se debe partir de un conjunto de patrones de entrenamiento de los que no conocemos su etiqueta de clase. Estos algoritmos se usan en situaciones de falta de conocimiento experto o cuando el etiquetado es totalmente impracticable. Aunque también se puede encontrar técnica de aprendizaje automático que usan ambos aprendizajes (redes bayesianas, redes neuronales, ..).

A continuación abordaremos las técnicas usadas según el algoritmo usado en el campo de la detección de intrusiones:

Inducción de reglas

Esta técnica para poder encontrar patrones secuenciales temporales en una secuencia de eventos. Así el sistema *Time-based Inductive Machine* (TIM), se descubrían los patrones que se encontraban en las anomalías del sistema. Por otra parte se ha usado a lo largo de la historia el algoritmo RIPPER es una aplicación usada para el aprendizaje o inducción de conjuntos de reglas sobre un conjunto de datos el cual se ha etiquetado con anterioridad. Aunque para datos con gran cantidad de ruido hay algoritmos mejores como han demostrado los arboles de decisión c4.5, a demás que genera reglas más sencillas de entender.

Uno de los trabajos más relevantes en la historia de los IDS fueron Lee y Stolfo [40] que propusieron el uso de RIPPER para la detección de anomalías las cuales generaba reglas automáticamente.

Árboles de decisión

Los árboles de decisión son un modelo que dada una base de datos construyen diagramas de construcción lógica, son muy similares a sistemas basados en reglas.

Otro trabajo fue Levin [41] que con la ayuda de la herramienta *Kernel Miner* desarrollo un conjunto de árboles de decisión de dónde se seleccionaban los más óptimos para definir las clases. Por otra parte se han propuesto usar arboles de decisión ID3 y algoritmos genéticos para la creación automática de reglas, con el objetivo el etiquetado de las conexiones y usar las reglas a través de un sistema experto.

Por otra parte una propuesta diferente sería la de crear un marco de dos etapas: uno de ellos basado en el aprendizaje de un modelo basado en reglas (PNrule) y otro desarrollado para aprender modelos de clasificadores en un conjunto de datos con distribuciones de clases muy distintas en el conjunto de entrenamiento.

Otro método es el basado en firmas de ataques como el que propone Nong [42] para aprendizaje en los árboles de decisión. Más adelante hizo pruebas con los algoritmos CHAID y GINI, a través de la herramienta *Answer Tree*. Obtuvieron resultados satisfactorios que nos indicaban que los arboles de decisión deben poder realizar un aprendizaje incremental pero asequibles y escalables para vastas cantidades de datos.

Al igual que anteriores trabajos, se podría usar la generación de firmas de ataques de forma automática en la búsqueda de anomalías en el tráfico con la finalidad de descubrir patrones y firmas de ataques. Para ello implementaría con *Signature A priori*, reglas de asociación y el algoritmo de árboles de decisión ID3, obteniendo una mejora de la eficiencia y la precisión en el descubrimiento de firmas.

Redes Neuronales

Las redes neuronales son sistemas basados en el aprendizaje y procesamiento automático, en el campo de la detección de intrusiones, proporcionan una alternativa flexible frente a los cambios que da el entorno, además de hacer frente a la búsqueda de ataques conocidos.

Uno de los primeros sistemas fue el de Debar y Dorizzi [43] usaron las redes neuronales para predecir el siguiente comando, basándose en un conjunto de comandos anteriores. El sistema de aprendizaje está implementado para que la salida que produce la tome para la siguiente iteración. En su trabajo tiene como objetivo identificar los datos que no se asemejan al uso legítimo del usuario.

Después Cansian et al.[44] desarrollaron un Sistema adaptativo de detección de intrusiones en el que obtenían patrones en el comportamiento anómalo usando redes neuronales.

Para terminar Ramadas et al. [45] desarrollo un sistema de detección de anomalías de red en el que se usaba un análisis estadístico para la detección de amenazas en vez los mapas autoorganizativos.

Algoritmos Genéticos

Los algoritmos genéticos son métodos adaptativos para la resolución de problemas de búsqueda y optimización, los cuales están basados en el proceso genético de los organismos vivos. En los sistemas de intrusiones se usa para optimizar el uso de características de los subconjuntos, haciendo los sistemas más precisos.

Uno de los primeros trabajos fueron los de Ludovic [46] que propuso este tipo de algoritmos para la búsqueda de posibles rastros de ataques. Más adelante presento su tesis y finalmente su proyecto utilizó un algoritmo genético para la búsqueda de ataques conocidos.

Los agentes son sistemas autónomos que realizan tareas en un entorno complejo y dinámico, se construyen a través de la programación genética y de una serie de funciones de los cuales disponen información de todo tipo.

También se han usados sistemas híbridos de algoritmos genético y arboles de decisión para la creación automática de reglas, donde la red y su comportamiento se traducen en reglas que puedan ser usadas como cromosoma de una población. De esta manera las reglas pueden ser usadas como base del conocimiento.

Por otra parte Helmer et al. [47] decidieron usar un conjunto de características y aplicar el algoritmo genético sobre estas para seleccionarlas y así reducir el número de características que se necesitan para la detección. Usando un clasificador de conexiones de red Li para distinguir entre el uso legitimo y normal, el algoritmo genético evolucionaba para generar un conjunto de nuevas reglas para el sistema de detección de intrusiones.

Clasificadores Bayesianos

Los clasificadores bayesianos se basan en los principios estadísticos y fundamentados por el teorema de Bayes para definir las clasificaciones probabilísticas. Estos sistemas han demostrado ser una herramienta muy útil en los sistemas de detección de anomalías en modelos de decisión.

Naive Bayes es un tipo de redes bayesianas que clasifica en un conjunto finito de clases uno conjunto de datos de entrada.

Uno de los primeros trabajos aplicando redes bayesianas proponía el uso de las redes con probabilidad condicional para ayudar al IDS. Una de la investigaciones más importantes fueron las de Valdes y Skinner [48] en su IDS EMERALD usaban las redes Bayesianas para el análisis de explosiones de tráfico.

Por otra parte, uno de los sistemas más interesantes aplicando redes bayesiana es en el cual se usa estimadores pseudo-Bayes para mejorar la precisión de su sistema de anomalías, también se hace uso de las reglas de asociación permitiendo reconocer los tipos de ataques en los cuales se habría entrenado previamente. Para clasificar los ataques se usaría naive Bayes donde los etiqueta como ataques nuevos, conocidos o normales.

Más adelante, se usaron las redes Bayesianas para determinar los eventos según la

información que emitían los diferentes modelos usados para la detección de anomalías.

Sistemas Inmunes Artificiales

Los sistemas Inmunes Artificiales presentan una interesante técnica basada en los conceptos inmunológicos. Ellos definen el conjunto de conexiones de lo normal tráfico como el 'yo', a continuación, generar un gran número de 'no-yo'. De esta manera el organismo trata de asegurar que los mecanismos de defensa que activen el sistema se enfoque contra los ataques.

Los sistemas inmunológicos permiten la detección distribuida y es una detección probabilística, a demás de entiempo real pudiendo reconocer cualquier tipo de anomalía. Al principio se usaban el modelo de selección negativa en la búsqueda de amenazas e ficheros.

Dos años más tarde Forrest et al. [49] desarrollo su trabajo de detección de intrusión a partir de las llamadas a los procesos UNIX. A partir de estos trabajos se desarrollaron trabajos para mejorar su sistema de detección de anomalías. En estos trabajos monitorizaban los rastros de llamadas al sistema y los comparaba con los datos de los que disponía almacenado de comportamiento normal de datos creado en el periodo de entrenamiento. Esto permitía al sistema detectar cualquier desviación del comportamiento normal y tratarla, para probarlo montó una red local y los detectores monitorizaban los paquetes TCP.

Por otro lado, se desarrollaron un sistema similar al de Forrest, Hofmeyr et al.[49] pero con un alfabeto mayor, cadenas diferentes y distintos umbrales. Al final concluyeron la imposibilidad del uso de este sistema para IDS, más adelante propusieron el uso de la selección clonal en vez de la negativa. Otra propuesta sería centrase en el análisis de las características que se seleccionan para su sistema y estuvieron comparando los dos sistemas anteriormente mencionado.

También usaron sistemas con selección negativa Dasgupta y González [50] para la detección de intrusiones, usaron los datos de DARPA de los laboratorios Lincoln y contrastaron el uso de selección positiva y negativa, concluyendo el correcto uso de la selección negativa.

Finalmente González et al. [50] usando una variante de la selección negativa llamada la selección negativa de valor real y al año siguiente usaron muestras positivas para generar las negativas las cuales se usaban para su algoritmo de clasificación.

Modelos de Markov

Los modelos de Markov son un modelo estadístico en el que el tipo de aprendizaje está basado en secuencias. Se pueden distinguir, entre otros, las cadenas de Markov y los modelos ocultos de Markov (HMM ó Hidden Markov Models). El objetivo es determinar los parámetros desconocidos a partir de los parámetros observables.

Inicialmente fueron propuestas por Nassehi [51] para la detección de anomalías, construyendo su cadena de Markov de tamaño unitario. Un año más tarde se propondría un sistema en el que se basaba en el modelo oculto de Markov para crear los perfiles de usuario.

k-means

K-means es un método agrupación que tiene como objetivo la partición en k grupos, este método crea una partición y se va iterando sobre esta hasta que se satisface el criterio de parada.

Los primeros en usar estas técnicas fue Portnoy et al. [52] , para ello agruparon los datos de comportamiento normales y de amenazas. Utilizaron una modificación de k-means y etiquetaban los grupos generados heurísticamente como ataques o uso legítimo.

k-NN- Vecino más cercano

El método k-NN vecino más cercano es un algoritmo para la clasificación supervisada que utiliza el aprendizaje basado en instancias. En este tipo de técnicas se almacenan los datos de entrenamiento y cuando aparece un nuevo objeto se busca el de mayor similitud para etiquetarlo. Para este tipo de técnica se pueden usar varios algoritmos ya sean basados en la distancia o en la densidad.

En el sistema de detección de intrusiones se han propuesto la utilización de la Shell de Unix para su IBL (*Instase Based Learning*). También se ha usado este tipo de clasificadores los trabajos de Chan [53] y otras técnicas basadas también en la distancia obteniendo resultados prometedores.

Otra forma donde se puede usar este tipo de técnicas es usando la función de densidad para la detección de las anomalías en red, siendo muy similar método K-NN. Para terminar, se han propuestos trabajos que a través del método del vecino más cercano compartido, ha permitido una mayor flexibilidad en agrupaciones con mucho ruido.

Support Vector Machine

La técnica de *Support Vector Machine* (SVM) se basa en el aprendizaje supervisado, donde dado un conjunto de objetos de entrenamiento, podemos clasificar las clases y entrenar una SVM para construir un modelo que prediga la clase de un nuevo objeto. Uno de los primeros trabajos que se basan en este tipo de técnicas utilizó SVM múltiple tanto para la detección de anomalías como para la identificación de ataques.

Finalmente Mukkamal et al. [54] usaron varios SVM para la identificación del tráfico legítimo y el resto de ataques obteniendo buenos resultados, más adelante propone el uso conjunto de SVM y redes neuronales.

Reglas de asociación

Las reglas de asociación se han utilizado extensamente en los sistemas de detección de intrusiones. Tiene como finalidad la agrupación de un conjunto de datos respecto a una serie de varias características, esto permite encontrar relaciones internas entre los datos de una misma conexión.

Uno de los trabajos más relevantes fue el de Lee y Stolfo[41] que usaban reglas de asociación para calcular un conjunto de patrones de episodios frecuentes. En el proyecto ADAM explicado con anterioridad usaban esta técnica para la detección de anomalías y luego clasificarlas en reglas de instancias normales o anómalas. Más adelante, usarán para la detección de anomalías reglas de asociación y episodios frecuentes.

Lógica Fuzzy

La lógica difusa o borrosa, también conocida como lógica difusa, es una lógica que nos permite procesar datos inciertos. Al contrario que la lógica booleana que sigue el principio bivalente o binario en el que la lógica solo permite dos estados posibles (verdadero y falso), la lógica difusa consigue tener infinitas degradaciones entre el valor verdadero y falso.

Esta técnica es una de las más indicadas para ser usadas por los sistemas de detección de intrusos ya que permite gran cantidad de características cuantitativas y además podemos definir un intervalo para indicar un valor.

Uno de los primeros trabajos en la seguridad informática que utilizaba esta técnica implementó un prototipo de IDS el cual detectaba no solo anomalías usando lógica fuzzy, también utilizó el uso indebido a través de técnicas tradicionales de sistemas expertos basados en reglas.

Los trabajos de Dipankar dieron resultados muy interesantes usando las técnicas de lógica fuzzy en [55] proponía un algoritmo genético para la creación de reglas fuzzy, para ello usaban los datos DARPA y demostraron tener buenos resultados al distinguir entre el tráfico normal y anómalo.

También se puede hacer uso de la lógica difusa para el motor de razonamiento de los IDS, obteniendo una mejora notable en la velocidad y en los costes de la detección. Por otra parte una propuesta interesante que se hizo fue la de bajar la cantidad de falsos positivos a través de sistemas fuzzy usando la información de varios IDS. Otra forma de usar la lógica difusa es utilizar el aprendizaje las redes neuronales y la lógica difusa para la detección de intrusiones basando en los patrones de ataque.

En el 2004, se empezaron a usar una matriz de semejanza fuzzy a través de las conexiones de red para su posterior agrupación. Por último, Guan et al. [56] también usaron la lógica difusa para la definición de comportamiento legítimo y anómalo muy similar a trabajos previos.

2.3.2. Minería de datos en la correlación de alertas

La correlación de alertas es uno de los temas más relevantes hoy en día, ya que actualmente se dispone de un gran abanico de sistemas detección de intrusiones y estos a su vez generan una enorme cantidad de alertas. Por lo tanto, se hace necesario reducir la tasa de falsos positivos y clasificarlos para poder agruparlo, y así ayudar al operador para que pueda tomar las medidas que tome oportunas.

Muchos de los trabajos de correlación se centran en juntar las salidas de los múltiples sensores en una única salida y así poder agruparlas para disminuir el número de falsos positivos en las alertas.

Uno de los primeros trabajos fueron los de Ye et al.[57] donde usaban varios IDS para analizar los mismos datos, en su sistema proponían el uso de la teoría de Dempster-Shafer para correlacionar las salidas que produce los IDS y luego agrupar los resultados a través de un sistema de redes Bayesianas.

Un año más tarde, Manganaris et al. [58] buscaban obtener los datos del comportamiento normal de las alarmas a través del uso de los distintos sensores. Aunque ese mismo año también se propuso un sistema de fusión de datos basados en varios sniffers distribuidos, ficheros logs, perfiles de usuario y más información del sistema.

Basándose en el uso de escenarios Dain y Cunninghan [59], en el cual usando minería de datos definían la probabilidad de permeancia a cada uno de los escenario. También usaron escenario Valdes y Skinner [60], que desarrollaron trabajos de correlación y creación de escenarios a partir de detectores probabilísticos en su IDS. Un año más tarde en proponen una correlación de alertas que se obtienen de IDS, servicios de autentificación y software antivirus y agregación basada en características comunes. Este último sistema dispone de una amplia similitud al que nosotros proponemos.

Siguiendo el estilo de los sistemas de correlación Debar y Wespi [61] también usaron un algoritmo para la correlación y agregación de alertas. Primero correlacionaba eventos en función de unas reglas preestablecidas y después se agrupaban para poder averiguar las intenciones del atacante.

En el mismo año Svensson et al. [62] propone en su trabajo un estudio de las alertas a través de redes neuronales, lógica fuzzy y lógica subjetiva, para ello obtenía la información de los sensores, de manera que solo se emitía juicios positivo o negativos de los datos.

En 2002, Ning y Xu [63] presentó un estudio experimental para adaptar las estructuras principales índices de memoria y mejoró las consultas a la base de datos para mejorar el análisis de las alertas. Tres técnicas se presentaron: estructura de hyper-alert, un índice de dos niveles, y ordenar la correlación.

Por otra parte, Cuppens et al. [64] ha sugerido tratar las amenazas como intentos de violación de seguridad, en la correlación de los objetivos de los ataques. Ning en [65] estudio como averiguar las estrategias de las alertas y medir su semejanza entre las secuencias de los ataques. Otra propuesta es el uso de los escenarios de ataque dándole un peso a cada uno de ellos y así mejorar el estudio sobre los escenarios.

Para el problema de los falsos positivos se ha propuesto el uso de minería de datos para agruparlos. Inicialmente intentaban identificarlos buscando el origen que los provoca para su futura eliminación, los resultados fueron positivos ya que consiguió reducir drásticamente la tasa de falsos positivos. Por otra parte se han desarrollado trabajos donde se proponía para el sistema de correlación de alertas uso de logs. Siendo posible dos modelos; uno de ellos analizaba el ataque para inferir su log y el segundo usa varios log para inferir las instancias del ataque. Después se correlacionan con árboles de decisión con otros logs.

Un año más tarde, Noel et al. [66] propuso grafos de ataques basados en asociación para correlacionar las intrusiones. En 2006 se propuso el uso de las estrategias de ataque y mediante técnicas probabilistas para el diseño del sistema de correlación.

Finalmente, Sadoddin y Ghorbani [67] en 2008 propuso un sistema de correlación en tiempo real compuesto por dos partes, una parte de agregación de alertas para los patrones estructurado y otra de minería para patrones frecuentes, en el cual nos fijamos para el desarrollo de nuestro sistema.

Actualmente no se disponen de trabajos de correlación relevantes ya que la tendencia actual en los sistemas de detección de intrusiones es el uso de sistemas distribuido y soft computing, en el cual los sistemas tratan de aprender y razonar en un entorno de incertidumbre e imprecisión. Estas técnicas abarcan una gran cantidad de metodologías (lógica fuzzy, sistemas inmunes artificiales, ...) los cuales trabajan de forma conjunta y cooperativa. Pero nosotros no hemos enfocado nuestra propuesta a sistemas distribuidos.

2.3.3. Lógica Fuzzy

La lógica difusa como hemos mencionado anteriormente nos permite el uso de varios estados. Por tanto, deberíamos hablar de lógicas difusas en plural, ya que estas son esencialmente lógicas multivaluadas (admiten varios valores de verdad) que extienden a las lógicas clásicas.

En un conjunto difuso, también observamos una función que asocia a cada objeto del universo un valor, pero esta vez comprendido entre el intervalo [0,1] según el grado de pertenencia del objeto al conjunto difuso. Las principales operaciones realizadas son:

- Fuzzification: Traducción de los valores del mundo real a valores difusos.
- Evaluación de reglas: Determinación de la fuerza de las reglas basadas en los valores de entrada y las reglas.
- Defazzification: Traducir de vuelta los resultados difusos a valores del mundo real.

Aplicaciones de la lógica difusa

Principalmente la lógica difusa está enfocada a la toma de decisiones cuando existen datos o conocimientos inciertos. Por otro lado, también es usada en el reconocimiento de patrones ambiguos o como un componente de sistemas expertos difusos. Ser la que se utilizará en este trabajo como clasificador de ataques y par del grado de pertenecía de patrones en un intervalo de valores dado.

Predicados borrosos

Los predicado borrosos son un conjunto de reglas que dividen un objeto en diferentes conjuntos borrosos, pero hay predicados que al aplicarlos a los elementos de un universo, no lo dividen perfectamente en dos subconjuntos, el de los que cumplen dicho predicado y

el de los que no lo cumplen. A este tipo de predicados se les denomina predicados borrosos o fuzzy.

El conjunto de los diferentes predicados nos definen el sistema difuso, donde cada uno de los predicados está formados por las funciones de pertenencia.

Funciones de pertenencia

La función de pertenencia de un conjunto nos indica el grado en que cada elemento de un universo dado, pertenece a dicho conjunto. Es decir, la función de pertenencia de un conjunto A sobre un universo X será de la forma: μ_A:X $\rightarrow$ [0,1], donde $\mu_A(x) = r$ si r es el grado en que x pertenece a A.
Si el conjunto es nítido, su función de pertenencia o función característica tomará los valores en (0,1), mientras que si es borroso, los tomará en el intervalo [0,1]. Si $\mu_A(x) = 0$ el elemento no pertenece al conjunto, si $\mu_A(x) = 1$ el elemento sí pertenece totalmente al conjunto. Las funciones de pertenencia son una forma de representar gráficamente un conjunto borroso sobre un universo. Las más relevantes son:

1. Función Triangular Definida mediante el límite inferior a, el superior b y el valor modal m, tal que $a < m < b$. La función no tiene porqué ser simétrica.

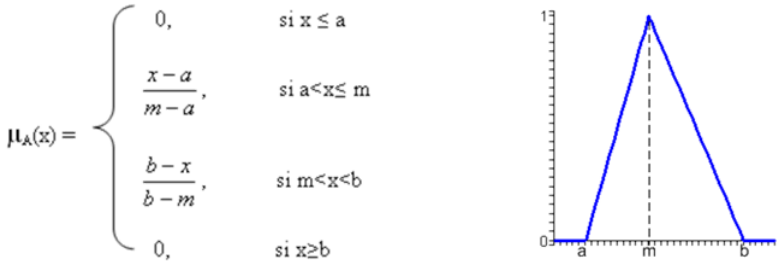

Figura 2.4: Función Triangular

2. Función Trapezoidal Definida por sus límites inferior a, superior d, y los límites de soporte inferior b y superior c, tal que $a < b < c < d$. En este caso, si los valores de b y c son iguales, se obtiene una función triangular.

3. Función Gamma Definida por su límite inferior a y el valor $k > 0$. Esta función se caracteriza por un rápido crecimiento a partir de a; cuanto mayor es el valor de k, el crecimiento es más rápido. Nunca toma el valor $\mu_A(x) = 1$, aunque tienen una asíntota horizontal en dicho valor.

Operaciones con conjuntos borrosos

Para la definición de los predicados borrosos se puede utilizar las distintas operaciones lógicas entre los conjuntos borrosos, a través de cada una de las funciones de pertenencia. Las más conocidas son:

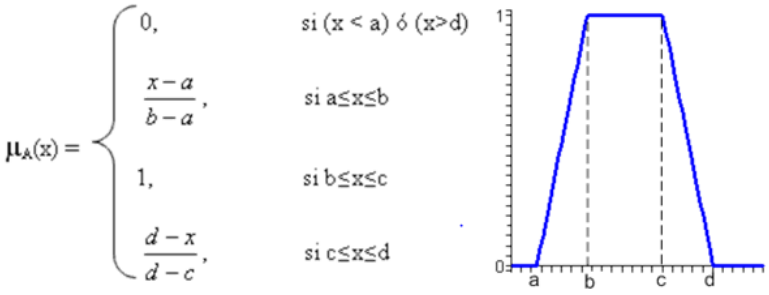

Figura 2.5: Función Trapezoidal

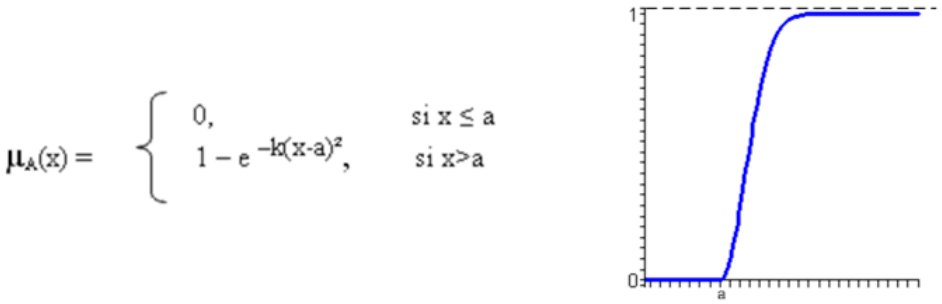

Figura 2.6: Función Gamma

- Intersección de conjuntos: El primer problema que nos planteamos es la obtención de la intersección de dos conjuntos borrosos. En el que dado dos subconjuntos P y Q, un elemento x pertenece a la intersección P∪Q, si y sólo si x pertenece a P y x pertenece a Q.
- Unión de conjuntos: Permite la unión de varios conjuntos borroso. En el que dado dos subconjuntos P y Q, un elemento x pertenece a la unión de P∩Q, si y sólo si x pertenece a P ó x pertenece a Q.
- Complemento de un conjunto: Finalmente, se puede realizar operación del complemento de un conjunto. Dado el subconjunto P, un elemento x pertenece al complemento Pc, si y sólo si dicho elemento x no pertenece a P.

2.3.4. Trabajos relacionados

Taxonomía de ataques

Una parte importante en la correlación de alertas es el tipo de clasificación en la cual se va a estructurar el sistema propuesto. Para ello es necesario categorizar los ataques e intrusiones.

Los primeros trabajos de clasificación en seguridad trataban de catalogar las debilidades de los sistemas informáticos y las vulnerabilidades funcionales, donde se clasificaban

y describieron unos 3000 tipos de ataques los cuales quedaban divididos en unas nueve categorías.

Kendall [68] desarrollo una base de datos de ataques, que mas tarde se encontraran en DARPA. Este sistema se defina en cuatro grupos según el tipo de ataque; denegación de servicio (DoS), exploración, R2L (*Remote to Local*, acceso a sitios sin permiso) y U2R (*User to Root*, el atacante dispone de cuenta e intenta atacar las vulnerabilidades).

En 2011 Hunt y Slay [69] presentaron un trabajo donde clasificaban los ataques en función de varias categorías; el sistema y la red, tipo de ataque, técnica de ataque y las tecnologías de protección.

Finalmente, una de las taxonomías completas que se puede encontrar es la que presenta el trabajo AVOIDIT [70], la que presentas distintas clasificaciones en función de (Vector de ataque, Impacto Operacional, Defensa, Impacto de la Información y objetivo). Una de las más completas y por la que al final nos decantamos fue la taxonomía en función del impacto operacional.

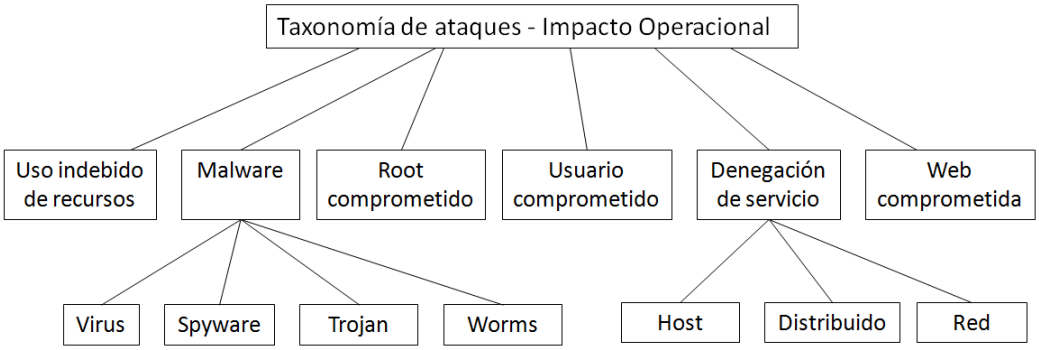

Figura 2.7: Clasificación de los ataques

La clasificación en función del impacto operacional, es decir consiste en catalogar los ataques según las consecuencias que estos ataques pueden provocar al usuario. A continuación exponemos los grupos en los que vamos a dividir cada categoría de ataque.

- Uso indebido de recursos
- Usuario comprometido
- Root comprometido
- Web comprometido
- *Malware*: (Virus, *Spyware*, *Trojan* y *Worms*)
- Denegación de servicio (Red, Host y Distribuido)

Capítulo 3

Sistema concurrente de detección de intrusiones con correlación de alertas en entornos distribuidos

El sistema que se propone en el presente documento busca dotar a una infraestructura de computación distribuida, en concreto de computación en la nube de una mayor seguridad y protección ante amenazas. Con este objetivo, en el presente capítulo se clasifica las distintas adaptaciones en tres grupos: propuestas de adaptación, paralelización del procesamiento del NIDS para aumentar su velocidad, y tratamiento de las alertas generadas. En el punto 3.1 se trata la definición de un escenario de pruebas, así como los lugares propuestos de despliegue para esta herramienta y se definen las amenazas contra las que deberá estar mejor entrenada. En el punto 3.2 se verán diferentes propuestas y pruebas para aumentar la velocidad de proceso de los paquetes de red basadas en aprovechar la paralelización tanto de la CPU como de la GPU. En el punto 3.3 se verá cómo agrupar las alertas generadas para simplificar la tarea de evaluarlas, y también servirá como método para reducir la tasa de falsos positivos del sistema.

3.1. Propuestas de adaptación de un NIDS a entornos distribuidos

Ya se han definido las características de la computación distribuida y de la computación en la nube. También se ha estudiado el crecimiento que ha tenido la computación en la nube y el crecimiento previsto. Se han enumerado las muchas ventajas que supone adoptar esta tecnología. Así mismo y debido a la novedad de dicha tecnología se han expuesto las vulnerabilidades a tener en cuenta según la *Cloud Security Alliance*. Visto el crecimiento de esta tecnología y las carencias que presenta en cuanto a la seguridad, se ha decidido lanzar una propuesta para añadir seguridad por medio de la adaptación de un NIDS existente. El NIDS que se va a adaptar es SDARP[5], que como se ha descrito anteriormente está basado en SNORT, añadiendo detección por anomalías del tráfico, analizando la carga útil de los paquetes de red.
Se procede a definir qué cambios mejorarían el comportamiento de los NIDS en entornos distribuidos, más concretamente en un entorno de computación en la nube. Las propuestas de adaptación se han clasificado en varios grupos:

- Lugares de despliegue del NIDS en la topología de la infraestructura de computación en la nube

  En este punto se definirá en qué nodos físicos del escenario será desplegado el NIDS, razonando los motivos de cada decisión. Para ello se definirán varios escenarios posibles y se verá si la misma propuesta se ajusta igual de bien a cada uno de ellos. Los lugares elegidos deberán cumplir varios requisitos, como permitir la escalabilidad del sistema, es decir, garantizar que no se penalizará el rendimiento de la infraestructura al escalar, y proteger adecuadamente los puntos más sensibles a las amenazas.

- Entrenamiento específico

  La propuesta, que consiste en desplegar varias instancias del NIDS por distintos puntos del escenario. Cada una de las instancias requerirá un entrenamiento específico orientado a detectar los posibles ataques ante los que esa parte de la arquitectura sea más vulnerable. La configuración de SDARP requiere de una etapa de entrenamiento, la cual nos permite perfilar el tráfico normal del sistema así como el anómalo para detectar los ataques con los cuales lo entrenemos. Para decidir las características de este entrenamiento se ha tenido muy en cuenta lo expuesto en el capítulo 2, de estado del arte, en las partes de riesgos de seguridad, y a la parte de trabajos relacionados.

- Aumento de la velocidad de proceso de paquetes de red

  Dadas las características de este paradigma de computación se requiere una mayor capacidad de proceso de paquetes que en una red local. Según la configuración del NIDS, éste puede estar analizando copias del tráfico de red (en tiempo real o no, para analizar si ha habido intrusiones) o el propio tráfico. En caso de que el análisis sea en tiempo real, una falta de velocidad provocaría o ralentizar la velocidad del tráfico de la red o dejar de monitorizar los paquetes que el NIDS no sea capaz de controlar. Esta propuesta se expone en el punto 3.2 del presente documento dada su extensión. En dicho punto se prueban varias alternativas de paralelización mixta entre la CPU y la GPU.

- Facilitar el tratamiento de las alertas generadas

  Ya ha sido comentado anteriormente que una de las características de los entornos distribuidos es que pueden escalar fácilmente, y que están más expuestos a amenazas. Debido al gran tamaño que pueden tener y a este riesgo añadido, es de suponer que el número de alertas crezca bastante. Sería de gran ayuda poder tratar estas alertas de manera que se pudieran priorizar las más urgentes y agruparlas para reducir su número. Los estudios en este campo son tratados ampliamente por separado en el punto 3.3 de este documento.

3.1.1. Escenarios

A la hora de hacer pruebas se contemplan arquitecturas pequeñas, de pocos nodos, suficientes para validar esta propuesta. Pero también se verá en este apartado escenarios con más nodos físicos para ver cómo se distribuyen los servicios del orquestador (el sistema que capacitará a las máquinas para ofrecer las funciones de una infraestructura de computación en la nube) sobre dichos nodos. Estas arquitecturas mayores servirán para validar en futuros trabajos uno de los requisitos que debería tener un NIDS que se quiere adaptar a una infraestructura de computación en la nube, que es el de no perjudicar el rendimiento del sistema al escalar éste. Pero de momento servirá para dar las explicaciones de por qué se confía en el cumplimiento de los requisitos de esta propuesta en estas infraestructuras con mayor número de nodos.
También se definirá en este apartado el sistema software con el que se dotará a la infraestructura de las capacidades de ofrecer servicios de computación en la nube, llamado orquestador o sistema operativo de computación en la nube. De los expuestos anteriormente en el estado del arte hemos elegido OpenStack [71]. La decisión se ha tomado por varios motivos. El primero es que es software libre, distribuido bajo la licencia Apache 2.0. Bajo esta licencia, su intención es crear estándares en este campo para que se pierda el miedo a las soluciones propietarias, que al no estar estandarizadas incrementan el coste de un cambio de plataforma.
En un principio fue desarrollado en 2010 por RackSpace hosting y la NASA, y más adelante, en 2011 adoptado por Ubuntu como su solución para ofrecer servicios en la nube, y desde sus inicios se han unido en su desarrollo más de 150 importantes empresas como Canonical, IBM, AT&T, HP, Dell, ... [72]. Todo este apoyo de la industria hace que sea una solución ampliamente aceptada. Este ha sido uno de los motivos, otro de ellos es que SDARP, el NIDS elegido, ha demostrado ser plenamente compatible con Ubuntu. Al estar OpenStack también plenamente probado con Ubuntu, se evitan problemas de compatibilidad y se facilita la instalación.

Estas son las tres arquitecturas propuestas para pruebas:

- Un solo nodo.

  En la figura 3.1 se muestra la instalación más básica con la que se puede probar OpenStack. Todos los componentes de OpenStack están instalados en un solo nodo, con lo que nos podremos hacer una idea del funcionamiento, pero no de su potencia.

- Dos nodos.

  En este caso todos los módulos de OpenStack se despliegan, como puede verse en la figura 3.2, en el nodo controlador salvo el módulo de computación, que se instala en el nodo de computación. Es la instalación mínima para producción. Aunque ambos nodos pueden funcionar en un entorno virtualizado, se recomienda ejecutar el nodo de computación en un nodo físico. Esta infraestructura se podría ampliar más adelante añadiendo nodos de cómputo.

- Más de dos nodos.

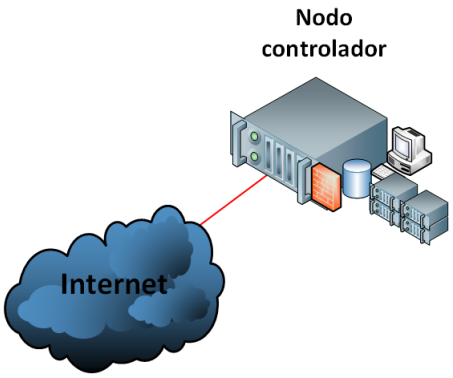

Figura 3.1: Arquitectura de despliegue en un solo nodo físico

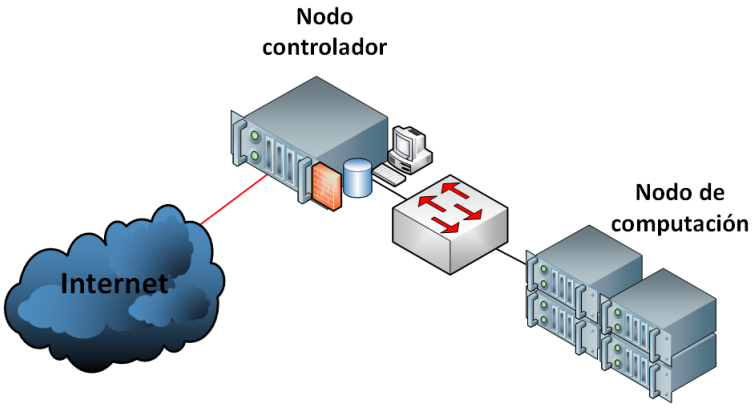

Figura 3.2: Arquitectura de despliegue en dos nodos físicos

La figura 3.3 ilustra el esquema de un entorno mínimo adecuado para producción. Tiene cuatro nodos. Uno de gestión de red (para gestionar la creación de redes virtuales complejas), otro de controlador, otro de computación y otro de almacenamiento. De cara a probar la efectividad del NIDS propuesto, en este proyecto se usará un escenario con un único nodo.

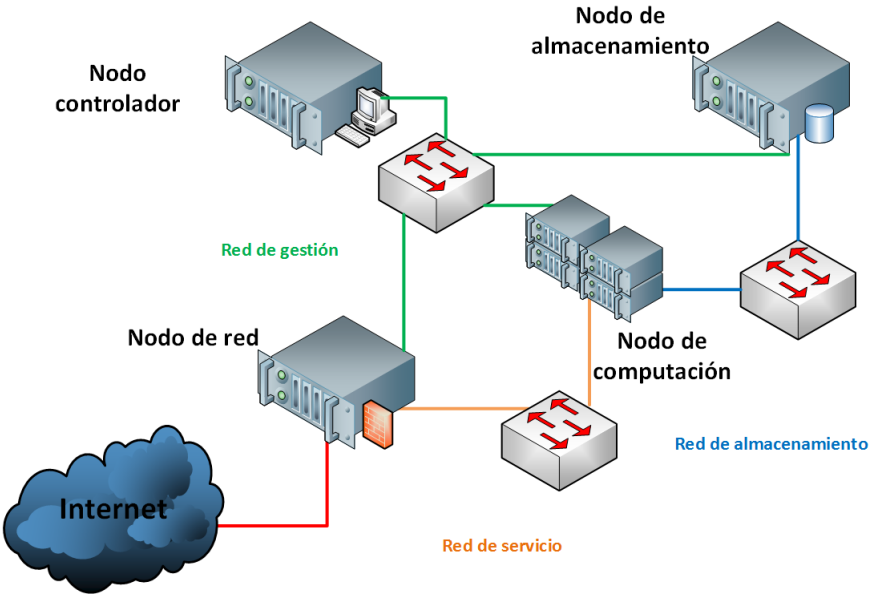

Figura 3.3: Arquitectura de despliegue en más de dos nodos físicos

3.1.2. Lugares de despliegue del NIDS en la topología de la infraestructura de computación en la nube

Una vez descritos los posibles escenarios, y elegido uno, el de un solo nodo con OpenStack, se procede a tomar la decisión de dónde desplegar el NIDS. Es cierto que en la arquitectura de un solo nodo no es muy complicado, ya que solo hay dos opciones, en el sistema operativo anfitrión o en cada máquina virtual, pero en este punto se expondrán las propuestas de ubicación en escenarios mayores enumerando las ventajas de las ubicaciones elegidas.

En el escenario de prueba, la ubicación más lógica es en el sistema operativo anfitrión. De esta manera se controlaría todo el tráfico entrante o saliente de la máquina, aunque no el que se pudiera producir entre las máquinas virtuales. Esto no es problema dado que al ser un entorno de prueba no hay posibilidad de que el número de máquinas virtuales sea muy alto. En este caso estarían protegidos contra ataques tanto los sistemas de identificación como los de gestión de máquinas virtuales, dos puntos críticos extraídos de los riesgos de seguridad expuestos en la sección de estado del arte.

En caso de escenarios con más nodos las ubicaciones propuestas serían dos. Se deberá situar un NIDS en la máquina que ejecute los servicios de controlador de la nube (nodo controlador), con el objetivo de asegurar este nodo ante amenazas contra los gestores de identidad de la nube. Estos servicios proporcionan autenticación a los usuarios, función muy importante dada la naturaleza multiusuario del sistema. Emplazar un NIDS aquí permitiría minimizar riesgos, por ejemplo de que se ejecute una *botnet* en la infraestructura como

consecuencia del robo de cuentas, o de ciertos tipos de denegaciones de servicio. En los mencionados escenarios con más nodos se propone desplegar un NIDS en el sistema operativo anfitrión (aquel sobre el que se ejecutan las máquinas virtuales, ejecutándose sobre la máquina directamente) de cada uno de los nodos de computación. El propósito de dichos nodos es albergar las máquinas virtuales que son ofrecidas a los clientes o usuarios de la infraestructura en la nube. Un ataque sobre este sistema operativo huésped pondría en peligro la confidencialidad, disponibilidad y la integridad de las máquinas virtuales contenidas en dicho nodo.
La elección de estos dos nodos (controlador y de computación) para desplegar los NIDS en la infraestructura no solamente está motivada por ser éstos puntos críticos de la misma. Eligiendo estos dos nodos se permite aumentar la seguridad del sistema sin perjudicar excesivamente el rendimiento. Cuando se hablaba de los requisitos que se imponían para adaptar un NIDS a la computación en la nube, uno de ellos era que este NIDS no impidiera al sistema de computación en la nube escalar, dado que esta es una de las principales características de las infraestructuras distribuidas. El despliegue de un NIDS en el nodo controlador de una infraestructura de producción con más de dos nodos no perjudicaría el rendimiento al escalar la misma añadiendo más nodos de computación o almacenamiento, ya que dicho NIDS no sería el encargado de filtrar el tráfico añadido por los nuevos nodos. Así mismo, como al añadir un nuevo nodo de computación se desplegaría un nuevo NIDS en su sistema operativo anfitrión, éste sería el encargado de filtrar el nuevo tráfico introducido al sistema. Estas razones permiten confiar en que el sistema NIDS propuesto cumple con el requisito de escalabilidad que se fijó en un principio.

3.1.3. Entrenamiento específico del NIDS

El NIDS que se usará para proteger un entorno distribuido, en este caso un entorno de computación en la nube es SDARP. Este NIDS como ya se ha comentado anteriormente lanza alertas basándose en la desviación del tráfico analizado del tráfico normal, lo que le permite detectar ataques de nueva creación, así como ataques polimórficos complementando a los sistemas basados en firmas de ataques conocidos. Para ofrecer esta protección y perfilar el tráfico legítimo e ilegítimo SDARP requiere de una serie de fases de entrenamiento. En este punto hablaremos de las amenazas más importantes contra los nodos del escenario que se van a proteger, el nodo controlador y los nodos de computación.

- Adaptaciones del NIDS del nodo controlador

  El nodo controlador contiene funciones valiosas que motivaron el hecho de situar un NIDS en este nodo. Una de dichas funciones es la gestión de identidad de los servicios de la infraestructura de computación en la nube. Otra función es la que provee a los usuarios de una interfaz web para gestionar sus recursos de computación, almacenamiento y gestión de redes. Como se comentará más adelante en la sección Apéndices, el servicio encargado de la autenticación de los servicios en OpenStack se llama Keystone, y el que provee de una interfaz web a los usuarios del sistema de computación en la nube se llama Horizon. El aprovechamiento de vulnerabilidades de estos servicios podría suponer el acceso a la gestión de los servicios de un usuario a un atacante, que a los ojos del sistema se convertiría en usuario legítimo. En este caso, el atacante podría tener acceso a los datos del usuario legítimo, y además ejecutar instancias cargando el coste a dicho usuario. Además una infraestructura

comprometida podría usarse para llevar a cabo ataques de denegación de servicio distribuidas.
Por lo expuesto anteriormente, el NIDS desplegado en el nodo controlador deberá llevar a cabo su entrenamiento con ataques de XSS (*cross-site scripting*), ataques de inyección de código cuyo objetivo es obtener privilegios por parte del atacante, ataques de desbordamiento de buffer, y ataques de inyección SQL [73].

- Adaptaciones del NIDS del nodo de computación

El nodo de computación es el nodo que se encarga de la creación y gestión de máquinas virtuales. Es por tanto un tipo de nodo que alberga la ejecución de los servicios de múltiples usuarios del entorno de computación en la nube. En este punto, los ataques más importantes contra los que deberá protegerse este nodo son ataques contra entornos virtualizados. Se entrenará el NIDS con tráfico con ataques de elevación de privilegios orientados al sistema operativo anfitrión, que es el que contiene el hipervisor o gestor de máquinas virtuales, entre ellos ataques de desbordamiento de *buffer*.
En [74] vemos un ejemplo de vulnerabilidad ante desbordamientos de *buffer* de un sistema de virtualización que permitía ejecutar código arbitrario. Cabe remarcar que este tipo de protección no es eficaz ante ataques entre máquinas virtuales ejecutándose en el mismo sistema anfitrión. Para vigilar este tipo de comunicaciones se recomienda usar un *VM firewall*, como los propuestos en [75]. Para el resto de las amenazas el NIDS propuesto supondría un gran beneficio en cuanto a protección.

3.2. Experimentos de paralelización de NIDS

Se han desarrollado diversas implementaciones de la fase de detección en C con la API de CUDA de NVIDIA y una en la que también se añade a la implementación paralelismo a nivel de CPU para obtener un mayor rendimiento. A continuación se explica y muestra los resultados obtenidos.

3.2.1. Escenario de Pruebas

Para el proyecto se ha hecho una nueva instalación con los componentes hardware y software que se muestran en las tablas 3.1 y 3.2. Se ha optado por realizar una instalación limpia y mínima para maximizar el rendimiento del hardware del que se ha dispuesto.

En la tabla 3.3 se muestran las especificaciones técnicas de la tarjeta gráfica empleada en las pruebas de rendimiento.

Para obtener los resultados de los distintos experimentos se ha realizado el análisis sobre un conjunto de paquetes de 187.7 kB.

CPU	Intel Core i7 2630QM
GPU	NVIDIA GEFORCE GT 520M

Tabla 3.1: Componentes hardware del entorno de desarrollo

SO	Ubuntu 12.04.1
Bumblebee version	3.2
CUDA toolkit version	4.1
Cuda drivers version	304.64

Tabla 3.2: Componentes software del entorno de desarrollo

3.2.2. Experimento 1

El objetivo de este primer desarrollo fue el de integrar la fase de detección, desarrollada en CUDA, dentro del NIDS. Para este primer experimento se optó por realizar la fase de ordenación de los cálculos relativos a la carga de datos de cada paquete mediante el algoritmo de ordenación burbuja porque CUDA no soporta algoritmos recursivos. El número de bloques y de hilos usados son establecidos al principio de la ejecución del programa. En la tabla 3.4 aparecen los resultados arrojados por este experimento y en la figura 3.4 se puede ver su evolución.

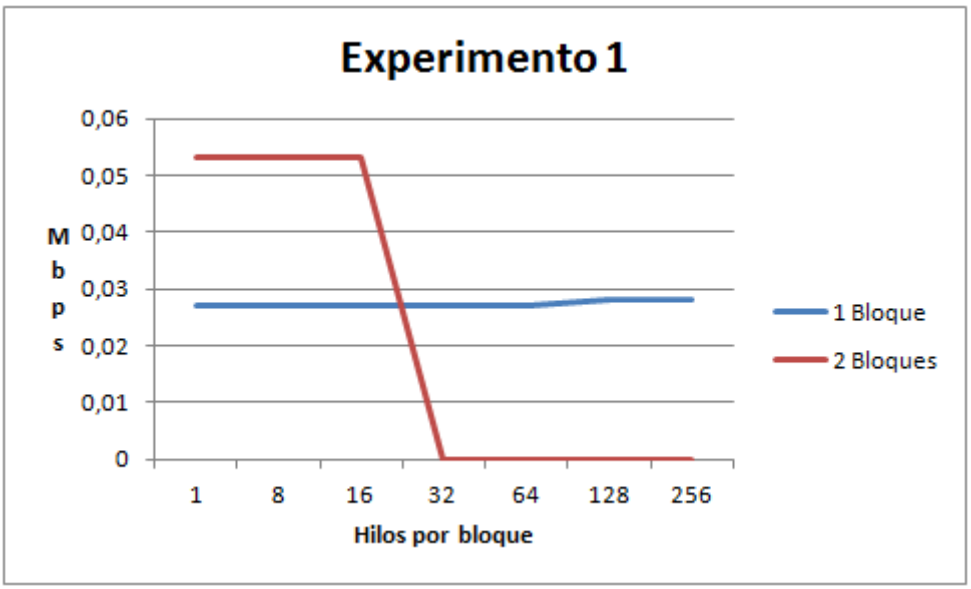

Figura 3.4: Gráfica con los resultados del experimento 1

Se puede observar como a medida que se aumenta el número de hilos también aumenta la velocidad de ejecución, la razón de este hecho es que cada hilo tiene que ordenar menos cálculos relativos a la carga de datos de cada paquete. También se puede ver como a partir de cierto número de hilos cuando se manda la detección usando dos bloques el tiempo de ejecución pasa a ser NaN, la razón de esto es que la tarjeta gráfica que se ha usado para estos experimentos no lo soportaba debido a que la lentitud de análisis hace que se

Compute capability	2.1
Clock rate	1480000
Total global memory	1073414144
Total constant memory	65536
Shared mem per multiprocessor	49152
Registers per block	32768
Warp size	32
Max threads per block	1024

Tabla 3.3: Especificaciones técnicas de la tarjeta gráfica NVIDIA GT 520M

le acumulen demasiados paquetes y se sobrepasan las capacidades de la GPU.

3.2.3. Experimento 2

El objetivo de este segundo experimento fue cambiar el algoritmo de ordenación en burbuja por el algoritmo de ordenación *quicksort* pasado a iterativo. La razón de este cambio fue la diferencia entre los órdenes de complejidad de estos algoritmos, en mucho menos tiempo podríamos realizar la fase de ordenación, con lo que se haría la fase de detección mucho más eficiente. En la tabla 3.5 aparecen los resultados arrojados por este experimento y en la figura 3.5 se puede ver su evolución.

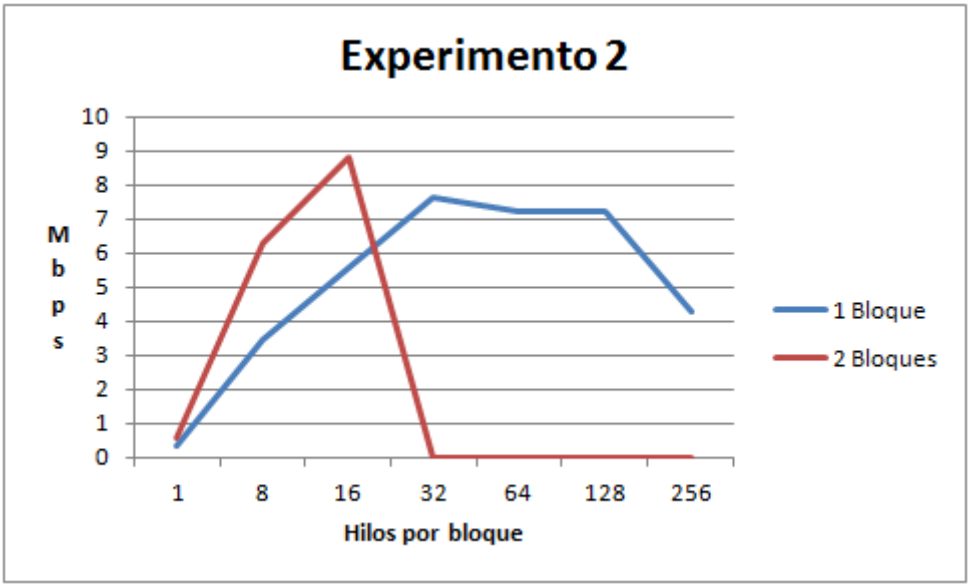

Figura 3.5: Gráfica con los resultados del experimento 2

Se puede observar una gran mejora respecto a los resultados del experimento 1, esto es debido a la diferencia en accesos a memoria que hay entre los dos algoritmos de ordenación. En las velocidades obtenidas se puede ver como cuando son lanzadas en un solo bloque la velocidad va en aumento hasta que se lanza con 32 hilos pero que a partir de ahí comienza a decrecer, la razón de esta curva es que, por un lado, por razones de eficiencia, es

Blocks per grid	Threads per block	Time (ms)	Speed (Mbps)
1	1	53699.30	0.027
2	1	27444.20	0.053
1	8	53638.14	0.027
2	8	27411.93	0.053
1	16	53639.17	0.027
2	16	27463.06	0.053
1	32	53694.49	0.027
2	32	NaN	-
1	64	52859.98	0.027
2	64	NaN	-
1	128	51819.97	0.028
2	128	NaN	-
1	256	51602.65	0.028
2	256	NaN	-

Tabla 3.4: Resultados del experimento 1

Blocks per grid	Threads per block	Time (ms)	Speed (Mbps)
1	1	4662.20	0.314
2	1	2464.36	0.595
1	8	425.19	3.449
2	8	234.36	6.257
1	16	263.44	5.566
2	16	166.19	8.823
1	32	192.80	7.605
2	32	NaN	-
1	64	203.18	7.217
2	64	NaN	-
1	128	202.61	7.237
2	128	NaN	-
1	256	343.60	4.26
2	256	NaN	-

Tabla 3.5: Resultados del experimento 2

aconsejable lanzar las ejecuciones con grupos de al menos 32 hilos, de ahí que hasta dicho número vaya en aumento, por otro lado al establecer de antemano este número de hilos y al analizar paquetes con tamaños distintos de carga útil se da el caso de que en algunos de ellos un número excesivo de hilos haga que solo unos pocos hilos estén realmente haciendo cálculos con lo que la activación/pausa del resto provoca una ralentización del sistema. También se puede ver como al igual que en el experimento 1 a partir de cierto número de hilos cuando mandamos la detección usando dos bloques el tiempo de ejecución pasa a ser NaN, la razón de esto es que, a pesar de que se ha conseguido una mejora en la velocidad no ha sido suficiente y la tarjeta gráfica continua sin soportar el análisis.

3.2.4. Experimento 3

A partir de los resultados obtenidos en los experimentos anteriores se pudo llegar a la conclusión de que la mayor parte del tiempo que tardaban ambos estaba dedicado a los algoritmos de ordenación, por lo que se optó por realizar los cálculos relativos a cada paquete en la GPU pero el resto de la fase de detección, incluyendo la ordenación de estos cálculos, hacerla en la CPU ya que los tiempos de intercambio de memoria en la CPU son menores que los de la memoria global de la GPU. También se decidió hacer el cálculo de bloques que se mandan a la GPU en función del tamaño del paquete a analizar para reducir al mínimo el número de cálculos que tiene que hacer cada hilo, ya que al tratarse de paquetes con cargas de datos de tamaño distinto, en muchos casos un número excesivo de hilos en cada bloque provocaría que muchos de estos no estuviesen realizando cálculos y que sobrecargasen al sistema con las llamadas de sincronización a nivel de bloque, como pudimos observar en los resultados arrojados por el experimento 2. En la tabla 3.6 aparecen los resultados arrojados por este experimento y en la figura 3.6 se puede ver su evolución.

Threads per block	Time (ms)	Speed (Mbps)
32	38.19	38.39
64	37.67	38.92
128	35.84	40.91
256	38.59	37.99
512	36.19	40.51
1024	36.42	40.26

Tabla 3.6: Resultados del experimento 3

Como puede observarse los tiempos se redujeron notablemente debido a que, por un lado, la GPU esta optimizada para realizar cálculos matemáticos pero en el aspecto de intercambios de memoria es muy lenta y por el otro, el uso de un número de hilos y bloques adecuado para cada paquete también le da al análisis una fluidez mayor.

3.2.5. Experimento 4

Teniendo como base los resultados obtenidos en el experimento 3, se decidió modificar la implementación para usar CUDA y OpenMP, por un lado la fase de cálculos de cada paquete se haría como en el Experimento 3, en la GPU, pero la fase de ordenación la

haríamos en la CPU con OpenMP. La razón de esto fue que el tiempo invertido por la CPU sin concurrencia se vería divido por el numero de hilos que usásemos con OpenMP. En la tabla 3.7 aparecen los resultados arrojados por este experimento y en la figura 3.7 se puede ver su evolución.

Threads per block	Time (ms)	Speed (Mbps)
32	27.3	53.71
64	26.92	54.47
128	27.49	53.34
256	27.51	53.30
512	28.53	51.39
1024	27.83	52.69

Tabla 3.7: Resultados del experimento 4

Respecto a los resultados obtenidos en el experimento 4 se puede observar una gran mejora en las velocidades obtenidas, esto se debe a que hacer uso de OpenMP permitió utilizar el mismo algoritmo de ordenación en paralelo para los dos vectores que teníamos que ordenar por cada paquete, lo que se tradujo en la división del tiempo empleado en la CPU sin concurrencia por un factor de dos.

También hay que tener en cuenta que se emplearon dos hilos para el uso del paralelismo en la CPU, ya que al tratarse de solo dos vectores para el mismo algoritmo era la mejor opción.

3.2.6. Resultados finales

Con el proyecto finalizado se procede a realizar una comparativa en la figura 4.1 entre la velocidad de análisis que se obtuvo en el proyecto realizado durante el curso 2011/2012 y la que se ha conseguido con estas investigaciones.

La representación que se ha hecho del proyecto SDARP en esta gráfica ha sido de una línea constante en los 12Mbps porque en él no influye el número de hilos que se emplean en la tarjeta gráfica.

En la tabla 3.8 se puede ver la ganancia obtenida en nuestros distintos experimentos respecto al proyecto SDARP y en la figura 3.8 una comparativa global.

Ganancia del experimento1	0,053/12	0,0044
Ganancia del experimento2	8,823/12	0,73
Ganancia del experimento3	40,91/12	3,4
Ganancia del experimento4	54,47/12	4,54

Tabla 3.8: Ganancias de los experimentos respecto a SDARP

Como se puede observar los resultados han sido muy favorables ya que en el mejor de los casos, que sería usando CUDA junto a OpenMP con 64 hilos, se ha conseguido una

Figura 3.6: Gráfica con los resultados del experimento 3

Figura 3.7: Gráfica con los resultados del experimento 4

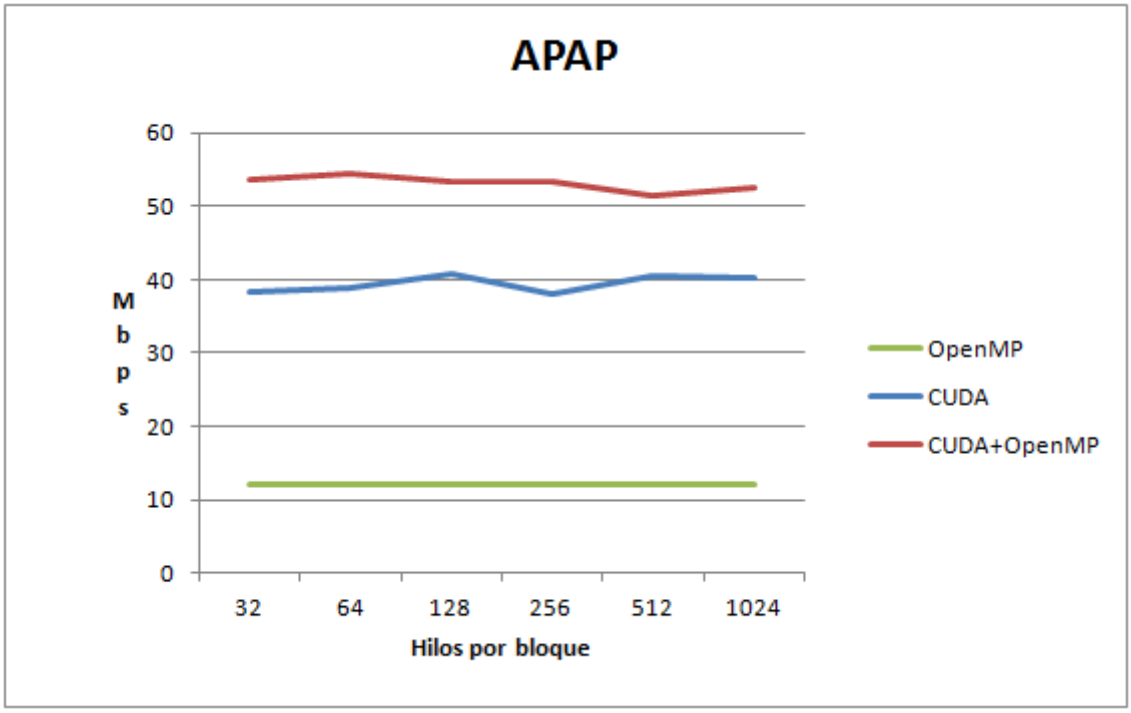

Figura 3.8: Comparativa de los resultados obtenidos en los experimentos

mejora de la velocidad del 354 %.

3.2.7. Comparación con NIDS comerciales

Para esta comparación se han utilizado exclusivamente los valores obtenidos en las métricas de rendimiento en la velocidad de proceso de los datos procedentes de la red.

Actualmente el mercado está mayoritariamente marcado por los NIDS por hardware. Esto es debido a que el rápido crecimiento de la velocidad de las redes implica una mayor velocidad de tratamiento de la información por parte de los NIDS y, a grandes escalas, no resulta viable la implementación por software.

En la tabla 3.9 se muestra una lista de productos NIDS en venta por los fabricantes mas punteros y trataremos de comprender hasta qué punto puede ser interesante el empleo de SDARP junto con nuestro algoritmo híbrido.

Se puede ver que este NIDS trabajando en un equipo con procesador Intel Core i7 2630QM y tarjeta gráfica NVIDIA GeForce GT 520M, alcanzando los 54'47Mbps, tendría un precio aproximado en el mercado de 2000 dólares.

El trabajar a dicha velocidad, sería una solución útil para redes de uso domestico y de pequeñas empresas, y probablemente funcionando bajo procesadores y tarjetas gráficas más recientes alcanzaría un rendimiento mayor.

3.3. Sistema de correlación de alertas

3.3.1. Sistema de correlación

La mayoría de los sistemas de correlación de alertas suelen tener una estructura parecida a la que se presenta en *A Survey of Alert Fusion Techniques for Security Incident*[76] y en este sistema se propone una estructura similar. Este NIDS se caracteriza por analizar la carga útil del tráfico monitorizado en busca de anomalías en el modo de uso habitual y legítimo de la red. Para ello cuenta con una fase de entrenamiento a partir de la cual genera un conjunto de reglas, que van a determinar los límites entre el tráfico legítimo y el tráfico anómalo. Cada regla es condicionada por un conjunto de valores que representan el contenido de la carga útil, en función de su frecuencia de aparición. Estos valores del espectro nos permitirán etiquetar los ataques. Antes de llevar a cabo el etiquetado, las alertas son normalizadas en base al estándar IDMEF [77].

3.3.2. Sistema de correlación de alertas fuzzy

Normalmente los sistemas de correlación de alertas se caracterizan por el tipo de información que usan véase cualitativo o cuantitativos, se propone usar ambas formas en la propuesta. El sistema de correlación de alertas que se propone se divide en dos etapas. Una primera etapa donde las alertas emitidas por SDARP son etiquetadas usando lógica difusa. Y una segunda etapa de agregación donde se aprovechan las características más representativas de las alertas y el etiquetado llevado a cabo en la etapa anterior.

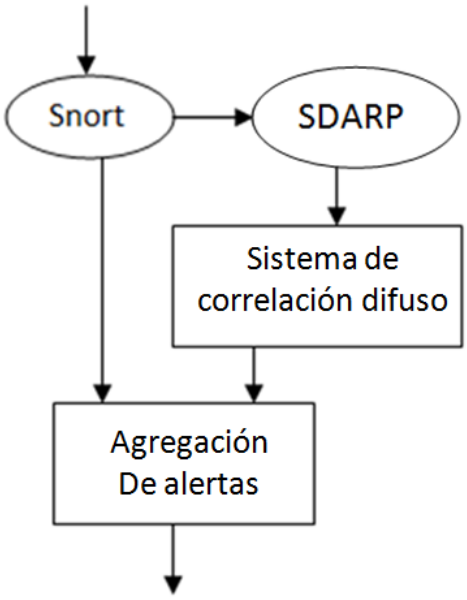

Figura 3.9: Arquitectura del sistema de correlación

En la figura 3.9 aparece el esquema general de la arquitectura donde podemos ver la integración de SDARP con Snort y como el sistema de correlación obtiene la información de las alertas de el NIDS, y por otro lado el sistema de agregación saca la información de

las alertas a través de Snort y de sistema difuso.

- Primera fase de correlación

  En la primera etapa se correlacionan las alertas que genera SDARP, para ello se usa la lógica difusa permitiendo la creación de una serie de reglas que definen un sistema difuso. Para implementar la lógica difusa que se ha definido se divide en tres partes como se observa en la figura 3.10.

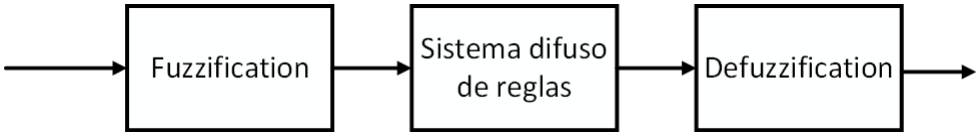

Figura 3.10: Lógica difusa

Fuzzification: Aquí se define las funciones de pertenencia que se obtiene de SDARP el valor y la frecuencia del espectro de la alerta.

Sistema difuso de reglas: Genera un sistema de regla y predicados borroso los cuales son los que clasifiquen las alertas según corresponda a un tipo de ataque.

Defazzification: Devuelve los datos difusos al lenguaje natural, el cual determina qué tipo de ataque es y el porcentaje de que pertenezca a ese tipo de ataques

Para el diseño de la correlación se ha querido hacer dos propuestas dependiendo de los datos que obtenemos del NIDS empleado. La primera de ellas, en la que se obtiene un conjunto de valores que indican el espectro de la carga útil del tráfico y su frecuencia de aparición. Al usar los valores del espectro para definir los conjuntos borrosos no solo indica si pertenece a un determinado tipo de ataque sino que permite saber el grado de pertenencia a este y así averiguar el porcentaje de que esa alerta pertenezca a un tipo de ataque. La segunda propuesta similar a la anterior donde se usa el conjunto de reglas que genera el NIDS junto a una base del conocimiento para poder etiquetar las alertas según el tipo de ataque.

En la figura 3.11 se puede ver cómo está estructurado el sistema difuso en función de los valores del espectro y su frecuencia de aparición.

Por otra parte sabiendo el grado de pertenencia del valor del espectro más significativo, esto ayuda a reducir el número de falsos positivos, descartando las alerta con un porcentaje de pertenencia realmente bajo. Actualmente se ha elegido el valor del espectro más significativo aunque se podría usar el resto de valores para averiguar si esa alerta podría ser de otro tipo y su grado de pertenencia a este. Para esta parte de la correlación se requiere tener una base del conocimiento previo que se obtiene a través del entrenamiento con SDARP, en esta parte se dispone de una colección de ataques y se estudia los resultados obtenidos del NIDS para poder ajustar el sistema

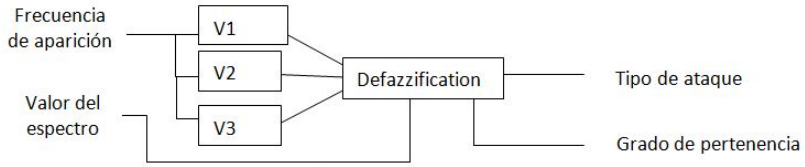

Figura 3.11: Arquitectura interna del sistema difuso

difuso.

- Segunda fase de agregación

  Por otra parte en la segunda etapa se ha usado un modulo de agregación donde se agrupan las alertas respecto a sus características, para ello se utiliza la asociación. En esta parte se utiliza la información obtenida del sistema de correlación para averiguar el tipo de ataque, la dirección IP del origen, la dirección de destino y el puerto. Para ellos se buscar si ya se dispone de una alerta con la misma serie de patrones para agruparla y, en caso de no encontrarla, añadiríamos un nuevo grupo según las características de asociación.

3.3.3. Lenguajes XFL

Actualmente existen varias herramientas para implementar la lógica difusa, los lenguajes XFL [139] nos permiten diseñar un sistema difuso fácil e intuitivo a través de XFuzzy. Una de las razones por la que se ha elegido esta herramienta es por la flexibilidad que proporciona tanto a la hora de diseñar las funciones de pertenencia como para realizar nuevas funciones, otra de las grandes ventajas es la posibilidad de migrar el sistema fuzzy a código c en el cual se basa todo el diseño del sistema de correlación de alertas. Las funciones de defuzzificación permiten obtener bastante información del sistema fuzzy, aunque para situaciones específicas se hace imprescindible crear otras que se adapten al sistema.

3.3.4. Resultados

Una vez concluida la parte de implementación se prueba el sistema de correlación, para ello se ha desarrollado un fichero donde se han almacenado todas y cada una de las alertas monitorizadas y detectadas por el sistema de detección de intrusiones. Este fichero tendrá todas las características necesarias para la futura correlación.

Se ha creado la prueba con 500 alertas simuladas que etiquetamos localmente en cada uno de los grupos de ataque. El sistema crea un fichero de salida en el que se muestran los tipos de ataques identificados y su grado de pertenencia a dicho conjunto de ataque. En este fichero ya esta aplicada la agregación por lo tanto solo nos aparecerán las distintas clases en las que se han agrupado. En la figura 3.12 podemos observar los tipos de ataques en los que están divididas las alertas que ha generado el sistema de correlación de alertas.

Modelo	Velocidad (Mbps)	Precio (Dolares)
Cisco 4210	40	500
Cisco 4270	4096	66000
PA-200	50	2000
PA-500	100	5200
PA-5xxx	2048	40000
Juniper IDP75	75	5000
Juniper IDP250	250	19000
Juniper IDP800	800	49000
Juniper IDP8200	8396,8	70000
HP J9521A	300	6000

Tabla 3.9: Comparativa con NIDS comerciales

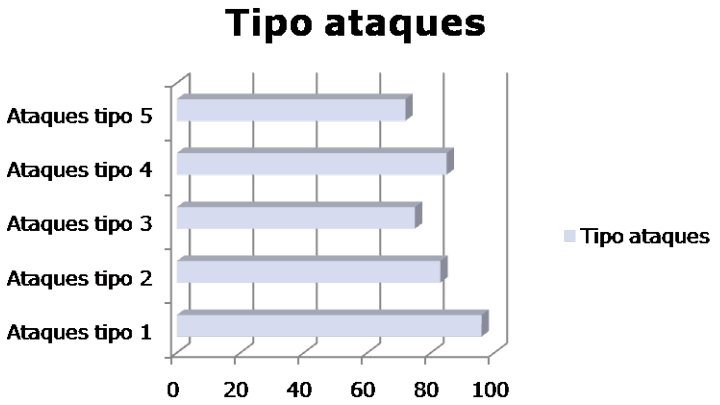

Figura 3.12: Clasificación de los ataques

Por otra parte en la figura 3.13 se puede ver del total de las 500 alertas el porcentaje de clases creadas, frente a las que han sido agrupadas.

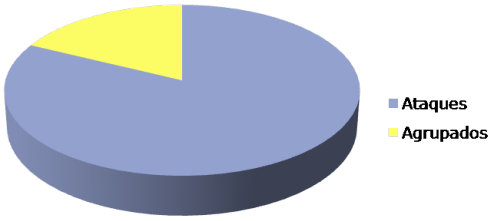

Figura 3.13: Agrupación de las alertas

Capítulo 4

Conclusiones y propuestas de trabajo futuro

4.1. Conclusiones

Con el proyecto Sistema Concurrente de Detección de Intrusiones con Correlación de Alertas en Entornos Distribuidos finalizado y tras exponer su contexto, bases y el trabajo realizado se procede a evaluar de manera crítica el trabajo realizado.

A *grosso* modo se opina que las metas fijadas al comienzo se han cumplido en su mayor parte de forma satisfactoria. La idea de adaptar un NIDS a entornos distribuidos, aumentar su rendimiento mediante la explotación de la concurrencia tanto a nivel GPU como CPU y crearle un sistema de correlación de alertas ha resultado un éxito.

Cabe destacar el trabajo de investigación realizado en temáticas totalmente desconocidas para el equipo de desarrollo. Este trabajo ha sido de gran valor cuando no indispensable para la consecución de los objetivos fijados. También hay que remarcar los resultados obtenidos en el apartado de rendimiento, ya que se ha conseguido una mejora en la velocidad, respecto al proyecto tomado como base, de un 354 % convirtiéndolo en un NIDS competente a nivel comercial y cuyo valor podría situarse en los 2000 dólares. Además del desarrollo satisfactorio de un sistema de correlación de alertas sólido, que ha demostrado tener resultado muy positivos.

En definitiva, se considera que el trabajo realizado ha merecido la pena y que gracias a esta investigación se ha conseguido convertir al prototipo tomado como base en una herramienta mucho más potente y robusta.

4.2. Propuestas para futuros trabajos

En este apartado se recogen los trabajos que se han considerado de interés por parte de los autores del documento para ser llevadas a cabo en el futuro. Se presentan tres propuestas: el desarrollo de un algoritmo de ordenación que haga uso de la concurrencia a nivel CPU para poder mejorar aún más el rendimiento del NIDS, el uso de los demás sensores de Snort para hacer al sistema de correlación de alertas mucho más preciso y ampliar el número de valores del espectro que usan para tener un mayor conocimiento de las alertas,

y la realización de pruebas del NIDS en los escenarios definidos de computación distribuida.

4.2.1. Algoritmo concurrente de ordenación CPU

En función de los resultados obtenidos a lo largo del desarrollo se ha podido ver cómo, a pesar de que gracias al empleo de concurrencia a nivel GPU se han obtenido grandes mejoras, el paralelismo en la GPU sólo era beneficioso si se usaba para realizar los cálculos referentes a cada paquete.

Por otro lado el uso de concurrencia a nivel CPU nos permitió obtener grandes mejoras de rendimiento en el apartado de ordenación de los datos devueltos por la GPU a pesar de que solo se emplease paralelismo a nivel de dos hilos. Es por esto que el desarrollo de un algoritmo que aprovechase un mayor número de hilos para realzar ordenaciones nos permitiría obtener un rendimiento mucho mayor.

4.2.2. Optimización del sistema de correlación de alertas

Aunque la propuesta actual sólo se centra en este NIDS, se puede usar en un futuro otros sensores que nos ofrece Snort tanto para mejorar el sistema con la información que ofrece el resto de preprocesadores como para ayudar en la detección de algunos ataques ya que por sus características no se pueden detectar.

Sería muy interesante usar el resto de valores del espectro de la carga útil para proporcionar más información sobre un tipo de alerta, lo que nos daría la posibilidad de darle más etiquetas para saber si es posible que pertenezca a más de un tipo.

4.2.3. Adaptación del NIDS a sistemas distribuidos

En el capítulo 3.1 se han definido las propuestas de adaptación de un NIDS para reforzar la seguridad en entornos de computación distribuida. Dichas propuestas han sido ideadas en base a los trabajos relacionados en ese campo, que debido a la novedad del mismo no son muy numerosos. Una de las propuestas de trabajo futuro sería seguir investigando en esta dirección, además de realizar pruebas en los escenarios definidos que respalden las adaptaciones propuestas en este documento.

Bibliografía

[1] Russell, Deborah, G. T. Gangemi. Computer security basics. O'Reilly Media, Inc., 1991.

[2] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, J. C. Phillips. GPU computing. Journals & Magazines, Vol. 96, pp. 879-899, Mayo 2008.

[3] T. Tamasi. Evolution of computer graphics. Proceedings of NVISION 2008, San José, California, Estados Unidos, Agosto, 2008.

[4] Nvidia, C. U. D. A. NVIDIA CUDA programming guide. (2011).

[5] Jorge Maestre Vidal, Hugo Villanúa Vega, José Ángel García Guijarro. Sistema de detección de anomalías de red basado en el procesamiento de la Carga Útil [Payload]. Proyecto de fin de carrera en Universidad Complutense de Madrid. Junio 2012.

[6] P. Mell, T. Grance. The NIST definition of cloud computing (draft). NIST special publication, 800, 145, Enero 2011.

[7] Forbes 2013 [online] disponible: http://www.forbes.com/sites/louiscolumbus/2013/02/19/gartner-predicts-infrastructure-services-will-accelerate-cloud-computing-growth/

[8] CSA [online] disponible: https://cloudsecurityalliance.org/download/the-notorious-nine-cloud-computing-top-threats-in-2013/

[9] The register [online] disponible: http://www.theregister.co.uk/2010/04/20/amazon_website_treat/

[10] The register [online] disponible: http://www.theregister.co.uk/2009/12/09/amazon_ec2_bot_control_channel/

[11] Cert definition of insider threat [online] disponible: http://www.cert.org/insider_threat/

[12] K. Vieira, A. Schulter, C. B. Westphall, C. M. Westphall. Intrusion detection for grid and cloud computing. It Professional, Florianopolis, Brazil, Vol. 12, No. 4, pp. 38-43, Agosto 2010.

[13] Ms. Parag K. Shelke, Ms. Sneha Sontakke, Dr. A. D. Gawande. Intrusion detection system for cloud computing. International Journal of Scientific & Technology Research Vol. 1, No. 4, Mayo 2012.

[14] S. Roschke, F. Cheng, C. Meinel. Intrusion detection in the cloud. Proceedings of the Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing, Potsdam, Germany, pp. 729-734, Diciembre 2009.

[15] Cert, denegación de servicio [online] disponible: http://www.cert.org/tech_tips/denial_of_service.html

[16] N. F. Huang, H. W. Hung, S. H. Lai, Y. M. Chu, W.Y. Tsai. A gpu-based multiple-pattern matching algorithm for network intrusion detection systems (Conference Publications). Proceedings of 22nd International Conference on Advanced Information Networking and Applications, Okinawa, Japon, pp. 62-67, Marzo 2008.

[17] GPGPU home page [online] disponible en: http://www.gpgpu.org

[18] AMD home page [online] disponible en: http://www.amd.com

[19] NVIDIA home page [online] disponible en: http://www.nvidia.com/content/global/global.php

[20] FireStream 9170 [online] disponible en: http://www.amd.com/es/products/workstation/graphics/legacy/Pages/firestream.aspx#3

[21] FireStream 9250 [online] disponible en: http://www.amd.com/es/products/workstation/graphics/legacy/Pages/firestream.aspx#2

[22] FireStream 9270 [online] disponible en: http://www.amd.com/es/products/workstation/graphics/legacy/Pages/firestream.aspx#1

[23] FireStream 9350 [online] disponible en: http://www.amd.com/es/products/workstation/graphics/legacy/Pages/firestream.aspx#4

[24] GeForce 8800 GTX [online] disponible en: http://www.geforce.com/hardware/desktop-gpus/geforce-8800-gtx/specifications

[25] GeForce GTX 280 [online] disponible en: http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-280/specifications

[26] GeForce GTX 480 [online] disponible en: http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-480/specifications

[27] GeForce GTX 780 [online] disponible en: http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-780/specifications

[28] Khronos home page [online] disponible en: http://www.khronos.org/

[29] Khronos Group. Khronos Launches Heterogeneous Computing Initiative (Press release), Junio 2008.

[30] Khronos Group. The Khronos Group Releases OpenCL 1.0 Specification (Press release), 8 Diciembre 2008.

[31] Khronos OpenCL API Registry [online] disponible en: http://www.khronos.org/registry/cl/

[32] OpenMP home page [online] disponible en: http://openmp.org/wp/

[33] C. Wu, J. Yin, Z. Cai, E. Zhu, J. Cheng. An Efficient Pre-filtering Mechanism for Parallel Intrusion Detection Based on Many-Core GPU. Proceedings of International Conference, SecTech, Held as Part of the Future Generation Information Technology Conference FGIT, Jeju Island, Korea. Lecture Notes in Security Technology, Vol. 58, pp. 298-305, Diciembre 2009.

[34] C. H. Lin, S. Y. Tsai, C. H. Liu, S. C. Chang, J. M. Shyu. Accelerating string matching using multi-threaded algorithm on gpu (Conference Publications). Proceedings of Global Telecommunications Conference (GLOBECOM, Miami, Estados Unidos, pp. 1-5, Diciembre 2010.

[35] G. Vasiliadis, S. Antonatos, M. Polychronakis, E. P. Markatos, S. Ioannidis. Gnort: High performance network intrusion detection using graphics processors. Proceedings of 11th International Symposium of Research in Attacks, Intrusions, and Defenses (RAID), Cambridge, MA, USA. Lecture Notes in Computer Science, Vol. 5230, pp. 116-134, Septiembre 2008.

[36] G. Vasiliadis, M. Polychronakis, S. Antonatos, E. P. Markatos, S. Ioannidis. Regular expression matching on graphics hardware for intrusion detection. Proceedings of 12th International Symposium of Research in Attacks, Intrusions, and Defenses (RAID), Saint-Malo, Francia. Lecture Notes in Computer Science, Vol. 5758, pp. 265-283, Septiembre 2009

[37] G. Vasiliadis, M. Polychronakis, S. Ioannidis. MIDeA: a multi-parallel intrusion detection architecture. Proceedings of the 18th ACM conference on Computer and communications security, Nueva York, Estados Unidos, pp. 297-308, Octubre 2011.

[38] K. Nordhaug. GPU Accelerated NIDS Search. Thesis for Master in Information Security. Department of Computer Science and Media Technology Gjøvik University College, 2012.

[39] O. Urko Zurutuza. "Sistemas de Detección de Intrusos" Universidad de Mondragón, España, pp. 1-47, 2004.

[40] W. Lee, S. Stolfo, k. Mok. Mining in a Data-flow Environment: Experience in Network Intrusion Detection. Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '99), San Diego, California, pp. 114-124, Agosto 1999.

[41] I. Levin. KDD-99 Classifier Learning Contest LLSoft's Results Overview. SIGKDD Explorations, Vol. 1, pp. 67-75, Enero 2000.

[42] X. Li, N. Ye. Decision tree classifiers for computer intrusion detection. Journal of Parallel and Distributed Computing Practices, Vol. 4, No. 2, pp. 179-190, 2001.

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

[44] J. Bonifacio, J. M., Cansian, A. M., Carvalho, E.S. Moreira. Neural networks applied in intrusion detection systems. Proceedings of The 1998 IEEE International Joint Conference on Neural Networks, Anchorage , Alaska, Vol. 1, pp. 205-210, Mayo 1998.

[45] M. Ramadas, S. Ostermann, B. Tjaden. Detecting Anomalous Network Traffic with Self-Organizing Maps. Web proceedings of the 6Th Recent Advances in Intrusion Detection RAID, Springer Berlin Heidelberg. Lecture Notes in Computer Science Vol. 2820, pp. 36-54, Septiembre 2003.

[46] L. Mé. A Genetic Algorithm as an Alternative Tool for Security Audit Trails Analysis. RAID International Symposium on Research in Attacks, Intrusions and Defenses, Bélgica ,pp. 14-16, Septiember 1998.

[47] G. Helmer, J. Wong, V. Honavar, L. Millee. Intelligent agents for intrusion detection. Proceedings of the Information Technology Conference, Syracuse, Nueva york, pp. 121-124, Septiembre 1998.

[48] A. Valdes, K. Skinner. Adaptive, model-based monitoring for cyber attack detection. Recent Advances in Intrusion Detection. Springer, Berlin Heidelberg. Lecture Notes in Computer Science Vol. 1907, pp. 80-93, Octubre 2000.

[49] S. Hofmeyr. An Immunological Model of Distributed Detection and its Application to Computer Security. Albuquerque, The University of New Mexico, 1999.

[50] F. González, D. Dasgupta. Anomaly detection using real-valued negative selection. Genetic Programming and Evolvable Machines, Vol. 4, No. 4, pp. 383-403, Diciembre 2003.

[51] M. Nassehi. Anomaly detection for Markov models. Technical Report IBM Research Division, Zurich Research, 1998.

[52] L. Portnoy, E. Eskin, S. Stolfo. Intrusion detection with unlabeled data using clustering. ACM Workshop on Data Mining Applied to Security (DMSA), Philadelphia, Pennsylvania, Estados Unidos, Noviembre 2001.

[53] P. k. Chan, M. Mahoney, M. Arshad. Learning Rules and Clusters for Anomaly Detection in Network Traffic. Managing Cyber Threats. Massive Computing Vol. 5, pp. 81-99, 2005.

[54] S. Mukkamala, A. H. Sung, A. Abraham. Intrusion Detection Using Ensemble of Soft Computing and Hard Computing Paradigms. Journal of Network and Computer Applications,Oklahoma, Estados Unidos, Vol. 28, no. 2, pp. 167-182, Abril 2005.

[55] J. Gómez, D. Dasgupta. Evolving fuzzy classifiers for intrusion detection. Proceedings of the 3Th Annual IEEE Information Assurance Workshop, Nueva York, pp. 321-323, Junio 2002.

[56] J. Guan, D. Liu, T. Wang. Applications of Fuzzy Data Mining Methods for Intrusion Detection Systems. Proceedings of International Conference on Computational Science and Its Applications (ICCSA). Assisi, Italia, pp. 706-714, Mayo 2004.

[57] N. Ye, J. Giordano, J. Feldman, Q. Zhong. Information Fusion Techniques for Network Intrusion Detection. Proceedings of the InformationTechnology Conference, Syracuse, Nueva york, pp. 117-120, Septiembre 1998.

[58] S. Manganaris, M. Christensen, D. Zerkle, K. Hermiz. A Data Mining Analysis of RTID Alarms. Computer Networks. Vol. 34, no 4, pp 571-577, Octubre 2000.

[59] O. Dain, R.K. Cunningham. Fusing a heterogeneous alert stream into scenarios. Proceedings of the ACM CCS Workshop on Data Mining for Security Applications, Philadelphia, Pennsylvania, Estados Unidos. Vol. 13, pp. 103-122, Noviembre 2001.

[60] P.A. Porras, M.W. Fong, A. Valdes. A Mission-Impact-Based Approach to INFOSEC Alarm Correlation. Proceedings of the Recent Advances in Intrusion Detection (RAID 2002), Springer Berlin Heidelberg, pp. 95-114, Octubre 2002.

[61] H. Debar and A. Wespi. Aggregration and Correlation of Intrusion-Detection Alerts. Proceedings of the 4th International Symposium on Recent Advances in Intrusion detection (RAID), Lecture Notes in Computer Science, pp. 85-103, 2001.

[62] H. Svensson, A. Josang. Correlation of Intrusion Alarms with Subjective Logic. Proceedings of the 6th Nordic Workshop on Secure IT systems. pp. 1-2, Copenhagen, Dinamarca, Noviembre 2001.

[63] P. Ning, D. Xu. Adapting query optimization techniques for efficient intrusion alert correlation, Technical report, NCSU, Department of Computer Science, 2002.

[64] F. Cuppens, A. Miege. Alert correlation in a cooperative intrusion detection framework. Proceedings of the 2002 IEEE Symposium on Security and Privacy, IEEE Computer Society, Berkeley, California, Estados Unidos, pp. 202-215, Mayo 2002.

[65] P. Ning, D. Xu, Learning attack stratagies from intrusion alerts. Proceedings of the 10th ACM Conference on Computer and Communications Security, Nueva York, Estados Unidos, pp. 200-209, Octubre 2003.

[66] S. Noel, E. Robertson, S. Jajodia. Correlating intrusion events and building attack scenarios through attack graph distances. Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC'04), Washington DC, Estados Unidos, pp. 350–359, Diciembre 2004.

[67] R. Sadoddin, A.A. Ghorbani. An incremental frequent structure mining framework for real-time alert correlation. Computers Security, New Brunswick, Canada, Vol. 28, No. 3, pp. 153-173, Junio 2009.

[68] K. Kendall, A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems, Tesis Doctotal, Massachusetts Institute of Technology Master, Massachusetts, Estados Unidos, Junio 1998.

[69] R. Hunt, J. Slay. A new approach to developing attack taxonomies for network security-including case studies. Proceedings of the 17th IEEE International Conference on Networks, Singapur, IEEE, pp. 281-286, Diciembre 2011.

[70] C. Simmons, S. Shiva, D. Dasgupta, Q. Wu. AVOIDIT: A cyber attack taxonomy. Technical Report. University of Memphis,Texas, Estados Unidos, 2009.

[71] OpenStack [online] disponible: http://www.openstack.org

[72] OpenStack companies [online] available: http://www.openstack.org/foundation /companies/

[73] N. Gruschka, M. Jensen. Attack surfaces. A taxonomy for attacks on cloud services. Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing, Miami, Florida, Estados Unidos, pp. 276-279, Julio 2010.

[74] VM firewalls [Online] disponible: http://www.securityfocus.com/bid/15998/info

[75] G. Young, N. MacDonald, J. Pescatore. Limited Choices Are Available for Network Firewalls in Virtualized Servers (Technical report). Gartner, 2007.

[76] T. Zang, X. Yun, Y. Zhang. A Survey of Alert Fusion Techniques for Security Incident. Proceedings of the 9th International Conference on Web-Age Information Management. Zhangjiajie, China, pp. 475-481, Julio 2008.

[77] D. Curry, H. Debar, B. S. Feinstein. The intrusion detection message exchange format (IDMEF). RFC 4765, IETF. Marzo 2007.

[78] D. López, F. J. Moreno, A. Barriga, S. Sanchez. XFL: a language for the definition of fuzzy systems. 6th IEEE International Conference on Fuzzy Systems. Vol. 3, pp. 1585-1591, Barcelona, España, Julio 1997.

[79] Bumblebee Project [online] disponible en: https://wiki.ubuntu.com/Bumblebee

[80] NVIDIA Developers [online] disponible en: https://developer.nvidia.com/

[81] Xfuzzy [Online] disponible: hhttp://www2.imse-cnm.csic.es/Xfuzzy/Xfuzzy_3.0/download_sp.html

Apéndices

Apéndices A

Adaptación de entorno a CUDA

A.1. Instalación de CUDA en Ubuntu 12.04.1

- Realizar todas las actualizaciones que nos pida el sistema, ya que de lo contrario durante la instalación de CUDA podrían darse ciertas incompatibilidades que hagan que se caiga el sistema.

- Instalar Bumblebee [79].

  Bumblebee 3.2 esta soportado por Ubuntu desde la versión 12.04 a la 13.04.

  Es necesario abrir una terminal como Administrador del sistema (root) e introducir los siguientes comandos:

  ```
  sudo add-apt-repository ppa:bumblebee/stable
  ```

  Para drivers más actuales es necesario añadir otro PPA. A partir de 12.04, esto sigue siendo necesario para la s tarjetas Nvidia GT 6xxM. Puede ser opcional para el GT y GT 4xxM serie 5xxM en 12.04. En caso de duda, simplemente instalarlo. El comando es:

  ```
  sudo add-apt-repository ppa:ubuntu-x-swat/x-updates

  sudo apt-get update
  ```

  Instale Bumblebee con el driver de NVIDIA:

  ```
  sudo apt-get install bumblebee virtualgl linux-headers-
     generic

  sudo reboot
  ```

- Instalar el cuda toolkit que soporte los drivers de nuestra tarjeta nvidia [80].
- Añadir a ".bashrc":

```
export PATH=$PATH:/usr/local/cuda/bin

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda/lib64
```

- Por ultimo instalar gcc version 4.4.x:

```
sudo apt-get install gcc-4.4

sudo apt-get install g++-4.4

sudo apt-get install build-essential

sudo update-alternatives \
    --install /usr/bin/gcc gcc /usr/bin/gcc-4.6 40 \
    --slave /usr/bin/g++ g++ /usr/bin/g++-4.6

sudo update-alternatives \
    --install /usr/bin/gcc gcc /usr/bin/gcc-4.4 60 \
    --slave /usr/bin/g++ g++ /usr/bin/g++-4.4
```

A.2. Guía de CUDA

La estructura de memoria que utiliza CUDA tiene una jerarquía de tres niveles, como se muestra en la figura A.1.

Memoria global: es la memoria utilizada para hacer lecturas o escrituras entre la CPU y la GPU, presenta el inconveniente de que los tiempos de intercambios de memoria son muy lentos.

Memoria compartida: es la memoria que comparten los hilos de un mismo bloque. Este tipo de memoria presenta unos tiempos de lectura/escritura más rápidos que la memoria global.

Memoria local: es la que tiene cada hilo que se lanza en una ejecución. Es la memoria con tiempos de lectura y escritura más veloces dentro la GPU, el inconveniente que tiene es que tiene un espacio muy reducido.

En la figura A.1 puede observarse la estructura de memoria en CUDA.

Para poder portar un programa escrito previamente en C a uno híbrido entre C y CUDA habría que hacer lo siguiente:

- Añadir al comienzo del Makefile la siguiente información:

```
# OS Name (Linux or Darwin)
OSUPPER = $(shell uname -s 2>/dev/null | tr [:lower:] [:
    upper:])
```

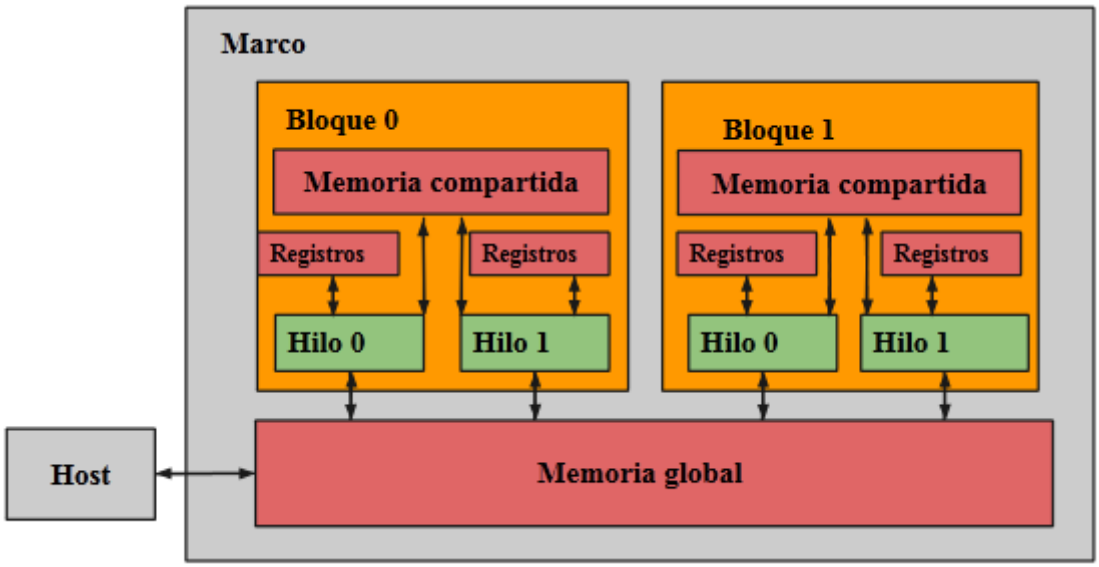

Figura A.1: Estructura de memoria en CUDA

```
OSLOWER = $(shell uname -s 2>/dev/null | tr [:upper:] [:
    lower:])

# Flags to detect 32-bit or 64-bit OS platform
OS_SIZE = $(shell uname -m | sed -e "s/i.86/32/" -e "s/
    x86_64/64/")
OS_ARCH = $(shell uname -m | sed -e "s/i386/i686/")

# These flags will override any settings
ifeq ($(i386),1)
        OS_SIZE = 32
        OS_ARCH = i686
endif

ifeq ($(x86_64),1)
        OS_SIZE = 64
        OS_ARCH = x86_64
endif

# Flags to detect either a Linux system (linux) or Mac OSX (
    darwin)
DARWIN = $(strip $(findstring DARWIN, $(OSUPPER)))

# Location of the CUDA Toolkit binaries and libraries
CUDA_PATH       ?= /usr/local/cuda
CUDA_PATH_LCUDA       ?= /usr/lib/nvidia-current
CUDA_INC_PATH   ?= $(CUDA_PATH)/include
CUDA_BIN_PATH   ?= $(CUDA_PATH)/bin
ifneq ($(DARWIN),)
```

```
  CUDA_LIB_PATH  ?= $(CUDA_PATH)/lib
else
  ifeq ($(OS_SIZE),32)
    CUDA_LIB_PATH  ?= $(CUDA_PATH)/lib
  else
    CUDA_LIB_PATH  ?= $(CUDA_PATH)/lib64
  endif
endif

# Common binaries
NVCC            ?= $(CUDA_BIN_PATH)/nvcc

# Extra user flags
EXTRA_NVCCFLAGS ?=
EXTRA_LDFLAGS   ?=

# OS-specific build flags
ifneq ($(DARWIN),)
      LDFLAGS   := -Xlinker -rpath $(CUDA_LIB_PATH) -L$(
          CUDA_LIB_PATH) -framework CUDA
      CCFLAGS   := -arch $(OS_ARCH)
else
  ifeq ($(OS_SIZE),32)
      LDFLAGS   := -L$(CUDA_PATH_LCUDA) -lcuda -L/usr/local/
          lib -lpcre
      CCFLAGS   := -m32
  else
      LDFLAGS   := -L$(CUDA_PATH_LCUDA) -lcuda -L/usr/local/
          lib -lpcre
      CCFLAGS   := -m64
  endif
endif

# OS-architecture specific flags
ifeq ($(OS_SIZE),32)
      NVCCFLAGS := -m32
           PTX_FILE        := spp_apap_kernel32.ptx
else
      NVCCFLAGS := -m64
           PTX_FILE        := spp_apap_kernel64.ptx
endif

# Debug build flags
ifeq ($(dbg),1)
      CCFLAGS   += -g
      NVCCFLAGS += -g -G
      TARGET    := debug
else
      TARGET    := release
```

```
endif

# Common includes and paths for CUDA
INCLUDES      := -I$(CUDA_INC_PATH) -I. -I.. -I../../common/
    inc -I../include
```

- Añadir al Makefile también las siguientes líneas, donde spp_apap_kernel.cu contiene las funciones de la GPU que serán llamadas desde el programa principal en C:

```
$(PTX_FILE): spp_apap_kernel.cu
        mkdir -p ./data
        $(NVCC) $(NVCCFLAGS) $(EXTRA_NVCCFLAGS) $(INCLUDES)
            -o $@ -ptx $<
        cp $@ ./data
```

- El archivo spp_apap.c es donde se realizarán las llamadas a las funciones de la GPU, para ello se tienen que poner las siguientes líneas:

```
#include <cuda.h>

#if defined(__x86_64) || defined(AMD64) || defined(_M_AMD64)
#define PTX_FILE "/home/proyectoSI/Descargas/snort-2.9.1.2/
    src/dynamic-preprocessors/apap/spp_apap_kernel64.ptx"
#else
#define PTX_FILE "/home/ proyectoSI/Descargas/snort-2.9.1.2/
    src/dynamic-preprocessors/apap/spp_apap_kernel32.ptx"
#endif
```

- Hay que definir las siguientes variables para poder realizar las llamadas al kernel correctamente:

```
CUdevice cuDevice;
CUcontext cuContext;
CUmodule cuModule;
CUresult error;
```

- Para poder realizar las llamadas a las funciones de la GPU es necesario inicializar las variables mostradas en el punto anterior. A continuación se muestra un ejemplo de función de inicialización:

```
bool inicializarVariablesCuda(){

        bool initCorrecto=true;
        int devID = 0;
        int deviceCount = 0;
        int i;
        bool sinErrores;
        error = cuInit(0);
```

```
        if (error != CUDA_SUCCESS){
                printf("cuInit fail\n");
                return false;
        }
        error = cuDeviceGetCount(&deviceCount);
        if (error != CUDA_SUCCESS){
                printf("cuDeviceGetCount fail\n");
                return false;
        }
        if (deviceCount == 0){
        printf("There is no device supporting CUDA.\n");
        return false;
        }
        if (devID < 0){
                devID = 0;
        }
        if (devID > deviceCount-1){
                return false;
        }
        int major, minor;
        char deviceName[100];
        error = cuDeviceComputeCapability(&major, &minor,
           devID);
        if (error != CUDA_SUCCESS){
                printf("cuDeviceComputeCapability  fail\n");
                return false;
        }
        error = cuDeviceGetName(deviceName, 256, devID);
        if (error != CUDA_SUCCESS){
                printf("cuDeviceGetName fail\n");
                return false;
        }
        error = cuDeviceGet(&cuDevice, devID);
        if (error != CUDA_SUCCESS){
                printf("cuDeviceGet fail\n");
                return false;
        }
        error = cuCtxCreate(&cuContext, 0, cuDevice);
        if (error != CUDA_SUCCESS){
                printf("cuCtxCreate fail\n");
                return false;
        }
        error = cuModuleLoad(&cuModule, PTX_FILE);
        if (error != CUDA_SUCCESS){
                printf("cuModuleLoad fail\n");
                return false;
        }
        return initCorrecto;
}
```

- Tras tener el entorno inicializado mostramos un ejemplo de cómo cargar una función de la GPU y almacenar memoria en memoria global de la tarjeta gráfica:

```
bool sinErrores;
//Cargamos la funcion
CUfunction funcion_gpu;
error = cuModuleGetFunction(&funcion_gpu, cuModule, "
    funcion_gpu ");
sinErrores = cudaCheckErrors(error);

//Reservamos memoria en el kernel
size_t size_double =sizeof(double);
CUdeviceptr ptr_double;
error = cuMemAlloc(&ptr_double, size_double);
sinErrores = cudaCheckErrors(error);
```

- Finalmente para hacer la llamada a la función de la GPU se hace lo siguiente:

```
void *args1[] = { &ptr_double };

//Launch the CUDA kernel
//blocksPerGrid es el numero de bloques que se usaran y
    threadsPerBlock es el numero de hilos que tendra cada
    bloque
error = cuLaunchKernel(funcion_gpu, blocksPerGrid, 1, 1,
    threadsPerBlock, 1, 1, 0, NULL, args1, NULL);
```

Para una información más detallada sobre el uso general de CUDA acudir a la guía oficial de nVidia [4].

Apéndices B

Adaptación Xfuzzy

El objetivo de esta sección es explicar los pasos a realizar a la hora de crear un sistema difuso con Xfuzzy.

B.1. Instalación Xfuzzy 3.0 en sistemas LINUX

Xfuzzy 3.0 es un entorno de desarrollo para sistemas de inferencia basados en La lógica difusa. Está formado por varias herramientas que cubren las diferentes etapas del proceso de diseño de sistemas difusos, desde su descripción inicial hasta la implementación final.

- Para poder ejecutar Xfuzzy es necesario disponer de cualquier plataforma de *Java Runtime Environment* (JRE). Para definir nuevos paquetes de funciones es también necesario disponer de un compilador Java. La última versión del *Java Software Development Kit*, incluyendo el JRE, un compilador Java y otras herramientas relacionadas.
- Descargar el fichero jar que podemos encontrar en [81].
- Extraer todos los ficheros y dejarlos en la carpeta donde deseemos trabajar
- Acceder a la carpeta donde se ha situado el programa a través del terminal, entrar en la carpeta "bin" y dar permisos al archivo "xfuzzy".

  Chmod 777 xfuzzy

- Ejecutar el archivo xfuzzy

  ./xfuzzy

B.2. Uso de XFL y Xfuzzy

Xfuzzy 3.0 es un entorno de desarrollo de sistemas difusos que dispone de varias herramientas para cada una de las diferentes etapas de diseño.

En la figura B.1 podemos ver la ventana principal donde podremos crear, modificar y verificar nuestros proyectos. Una vez creado nuestro nuevo proyecto a través de los menús básicos de la ventana lo editaremos, para ello solamente habrá que seleccionarlo.

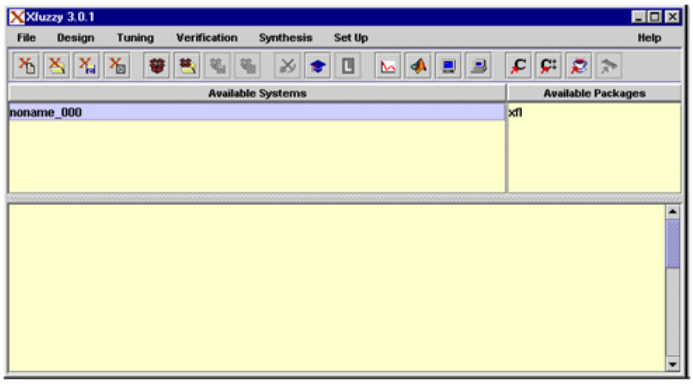

Figura B.1: Ventana principal

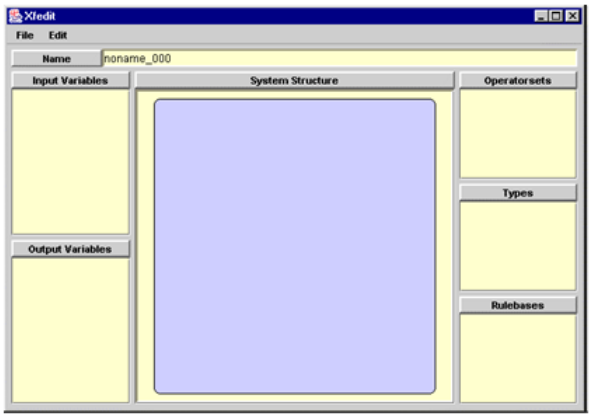

Figura B.2: Ventana de diseño

En la figura B.2 vemos la ventana donde podremos editar y desarrollar nuestro sistema difuso para ello seguimos una serie de pasos:

1. Definimos nuestras funciones de pertenencia, para ello seleccionamos el marco Types y pulsamos la tecla “Insert”.

   Aparecerá un menú como en la figura B.3, donde seleccionaremos las características que deseemos (para que funcione bien este paso es necesario poner el valor 255 en “cardinality” y poner al menos una función de pertenencia en “No, MFS” dándole un valor de 1 o superior. Una vez realizado esta parte ajustaremos la función de pertenencia en función de nuestras necesidades como aparece en la figura B.4

2. Definimos las variables de entradas y salidas del sistema difuso en función de los tipos que hemos creado en el paso previo. Para ello daremos a “Insert” en cada una de las respectivas casillas y las completaremos según los datos que hayamos seleccionado.

3. Creamos los operadores a través del marco “operatorSet”, donde nos aparecerá una ventana como en la figura B.5.

   Aquí seleccionaremos como queremos que sean las reglas de nuestro sistema difuso (and, or, . . .), un de las partes más importante es la defuzzification donde en nuestro

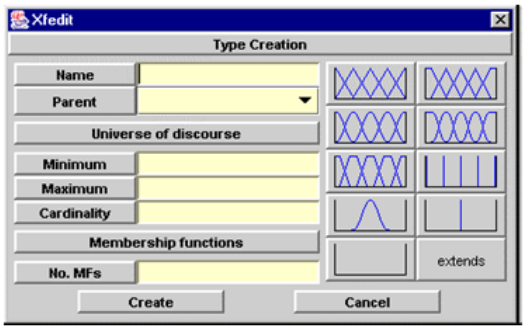

Figura B.3: Tipo de datos

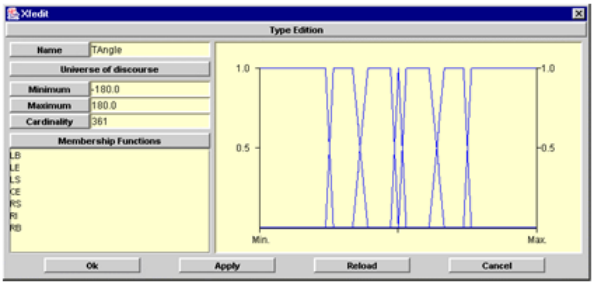

Figura B.4: Función de pertenencia

sistema se ha elegido MAX label, aunque para el sistema de defuzzification basado en el grado de pertenencia se recomienda dejar en blanco y ya se modificara código más adelante.

4. Por último seleccionamos la opción de RuleBase donde definimos las reglas en función de los operadores anteriormente definidos y las funciones de pertenencia como aparece en la figura B.6.

   Una vez definido solo tendremos que crear una llamada a la base de reglas a través de los menús básicos y finalmente unir todas las partes del sistema difuso tal y como corresponda.

Por otra parte para verificar nuestro sistema se aconseja seleccionar el menú verificación en la ventana principal y seleccionar monitorización. A través de esta ventana podemos verificar nuestros resultados como aparece en la figura B.7.

Finalmente en el menú principal seleccionamos “Synthesis” y elegimos código c para poder usar nuestro sistema difuso e integrarlo en nuestro trabajo. Para su correcto funcionamiento habrá que modificar el código basándonos en el trabajo ya realizado y se añadirán y sustituirán las partes que se han indicado en nuestro código.

Figura B.5: Operaciones difusas

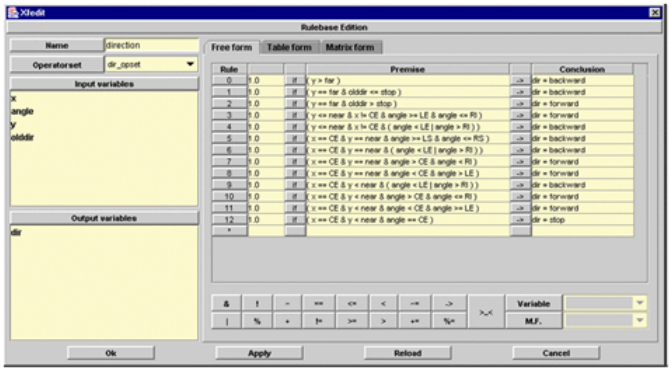

Figura B.6: Sistema de reglas

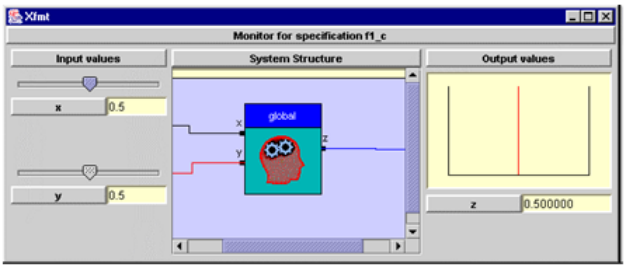

Figura B.7: Verificación

Apéndices C

Componentes de OpenStack y herramientas para su despliegue

En esta sección se dará más información acerca de los componentes que forman *OpenStack* y las distintas herramientas que se pueden usar para hacer un despliegue de pruebas.

C.1. Descripción y componentes de OpenStack

OpenStack es un proyecto *software* de código abierto cuyo objetivo es desplegar un sistema de computación en la nube en su modelo de servicio de infraestructura como servicio (*IaaS, Infrastructure as a Service* por sus siglas en inglés). Múltiples compañías del sector de las tecnologías se han unido al proyecto postulándose *OpenStack* como una firme candidata a servir de estándar para la computación en la nube a medio plazo.
El modelo de servicio *IaaS*, como se comentó en el capítulo 2, tiene como función ofrecer servicios de procesamiento, almacenamiento y gestión de redes bajo demanda del usuario desde un centro de datos. *OpenStack* está formado por una serie de componentes software interrelacionados con el objetivo de ofrecer los servicios anteriormente mencionados. Estos componentes básicos son siete:

- Almacenamiento de objetos (*Swift*)
  Swift es un sistema escalable para almacenar objetos. Se encarga de copiar objetos y ficheros repartiendo la información replicada entre los distintos servidores del centro de datos. Es resistente a fallos y fácilmente escalable añadiendo nuevos nodos de almacenamiento.

- Imágenes (*Glance*)
  La función de *Glance* es la de hacer de repositorio de imágenes de máquinas virtuales. Se encarga de proveer las imágenes de máquinas virtuales que ejecutará *OpenStack Compute*. También puede ser usado para realizar copias de seguridad de máquinas virtuales.

- Computación (*Nova*)
  Provee servidores virtuales bajo demanda. Es la parte principal a la hora de ofrecer el servicio de *IaaS*. Está diseñado para escalar horizontalmente, es decir, para poder ser ampliadas sus capacidades y acomodarse fácilmente a dicho crecimiento.

Automatiza la gestión de recursos de computación pudiendo trabajar con múltiples tecnologías de virtualización, tales como *XenServer* y *KVM*.

- Interfaz de gestión (*Horizon*)
  Ofrece una interfaz web para la gestión de todos los servicios de *OpenStack*, tanto a administradores como a usuarios. Mediante esta interfaz el usuario puede llevar a cabo la mayoría de las operaciones de gestión del sistema de computación en la nube, tales como lanzar una instancia. Esta es solo una manera de interactuar con el sistema. Para automatizar las gestiones con el sistema de computación en la nube (que se recuerda que era una de las características de la computación en la nube), se usa la *API* nativa de *OpenStack* o la de compatibilidad con Amazon EC2, que pese a ser una *API* de un sistema privado, no estándar, está ampliamente extendida y por ello soportada por *OpenStack*.

- Gestión de identidad (*KeyStone*)
  Se encarga de gestionar la autenticidad y los permisos de todos los servicios de *OpenStack*.

- Gestión de redes (*Quantum*)
  Quantum se encarga de proveer conectividad de red como servicio entre interfaces de red gestionadas por otros servicios (típicamente *Nova*). Este servicio ofrece a los usuarios la posibilidad de crear sus propias redes virtuales conectando distintos servidores virtuales del usuario. Esto permite disfrutar de las ventajas que ofrecen las redes definidas por *software* (*SDN, software defined network*), que están ganando popularidad últimamente.

- Almacenamiento de bloques (*Cinder*)
  Gestiona el almacenamiento a nivel de bloque para las instancias de computación de *OpenStack*. Se encarga de crear, vincular y desvincular dispositivos virtuales de almacenamiento de bloques a los servidores.

Estos han sido los siete servicios básicos que componen *OpenStack*.
La función de *OpenStack* en conjunto es la de servir como un sistema operativo de un sistema de computación en la nube. Para ello, todos estos componentes están diseñados para ofrecer juntos una infraestructura como servicio. Esta integración se lleva a cabo por medio de *API's* que cada servicio ofrece (y a la vez usa). Conceptualmente las relaciones entre los servicios quedan como se muestra en la figura C.1:

Toda la arquitectura explicada pertenece a la versión *Folsom* de *OpenStack*. En el momento de ser escritas estas líneas se publica la siguiente versión, *Grizzly*.

C.2. Herramientas para hacer un despliegue de pruebas

La complejidad para efectuar un despliegue de pruebas depende mucho de qué arquitectura se desee crear. En el apartado 3.1.1 de esta memoria se proponen tres arquitecturas

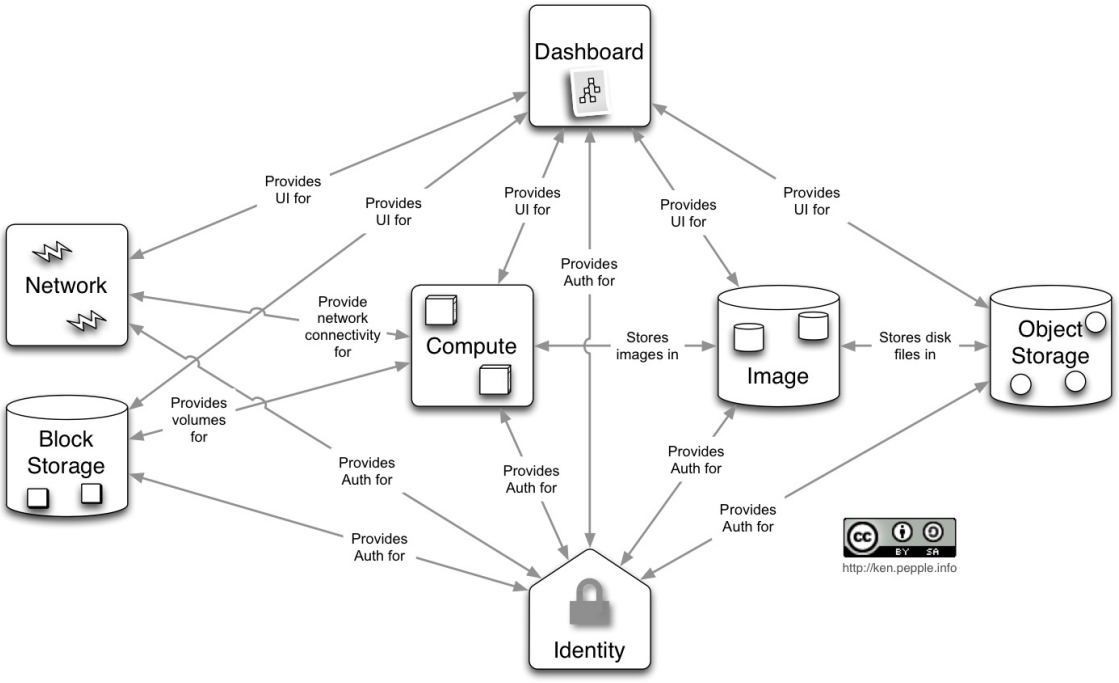

Figura C.1: Arquitectura conceptual de OpenStack

de prueba, de las cuales, las dos primeras no presentan complicaciones en su despliegue. Como parte de este proyecto se ha investigado la opción más fácil y útil a la vez que nos permitiera crear un escenario de pruebas, y finalmente se han elegido dos alternativas, cada una de ellas con sus ventajas y sus desventajas.

- *DevStack*
  DevStack es una solución independiente de *OpenStack* que ayuda a crear una infraestructura para pruebas con OpenStack. Esta solución se basa en una serie de *scripts* y de ficheros de configuración que automatizan la instalación de OpenStack sobre una instalación de los sistemas operativos soportados, entre ellos *Ubuntu server 12.04 LTS*, aunque también puede funcionar en otras distribuciones de *Linux*. En su página web, http://devstack.org, proponen cuatro opciones para probarlo. La primera opción posibilita instalar *OpenStack* en una máquina virtual, lo que facilita la tarea en caso de fallos y también en caso de querer desinstalar la herramienta, que será tan fácil como borrar la máquina virtual. Esta opción tiene como contra que no se puede probar la potencia de la herramienta ya que al estar funcionando en un entorno virtualizado el rendimiento se ve afectado negativamente.

  De las otras opciones nos interesan las dos primeras, que permiten desplegar *OpenStack* sobre una o varias máquinas físicas, ya sean servidores o portátiles. En ambos casos se puede valorar la potencia de la herramienta mejor que en la opción virtualizada, y en la opción multinodo se puede observar la distribución de los componentes de *OpenStack* entre las máquinas.

- *StackOps*

StackOps es una distribución de *Linux* (basada en *Ubuntu Server*) personalizada para poder llevar a cabo la instalación de *OpenStack* de manera remota desde otro ordenador en la misma red mediante una conexión *http*. El resultado en principio es el mismo que usando *DevStack* pero con muchas más opciones de personalización. Esto facilita enormemente la creación de arquitecturas de prueba de más de un nodo, ya que el primer nodo en el que se instala hay que elegir que sea nodo controlador, y en el segundo y sucesivos nodos de computación en el asistente web. Para más información acudir a la web www.stackops.com.

Como opinión personal de los autores, para crear un escenario de prueba de un único nodo muy rápidamente se recomienda usar una máquina virtual con *DevStack*, pero para probar en máquinas físicas con más nodos los autores recomiendan el uso de *StackOps*.