UNIVERSIDAD DE MURCIA
Facultad de Informática

PROYECTO FIN DE CARRERA

ARQUITECTURA DIRIGIDA POR EVENTOS PARA UN SISTEMA DE DETECCIÓN DE INTRUSIONES BASADO EN PATRONES

Autor
D. Jesús Joaquín Martínez Molina

Directores
D. Miguel Ángel Hernández Ruiz
D. Manuel Gil Pérez
Dr. Gregorio Martínez Pérez

Murcia, Septiembre de 2008

Índice

1. Introducción **1**

2. Análisis **4**
2.1. Sistemas de Detección de Intrusiones 4
2.1.1. Prelude 8
2.1.2. Snort 9
2.2. Bases de Datos de Vulnerabilidades 13
2.2.1. OSVDB (Open Source Vulnerability Database) 13
2.2.2. NVD (National Vulnerability Database) 18
2.3. Eventos 20
2.3.1. Definición de eventos de ataques 20
2.4. Técnicas de Correlación 22
2.5. Intrusion Detection Markup Language (IDML) 26

3. Diseño y resolución del trabajo realizado **28**
3.1. Arquitectura del Sistema 29
3.1.1. Capa de Preprocesamiento 29
3.1.2. Capa de Detección de Eventos 35
3.1.3. Capa de Decisión 45
3.2. Resumen de la Arquitectura 45
3.3. Ejemplo ilustrativo 46
3.4. Implementación del Prototipo 51

4. Conclusiones y vías futuras **53**

A. Reglas en Snort. Significado, formato y campos **57**
A.1. Introducción 57
A.2. Cabecera de las reglas 57
A.3. Opciones de la Reglas 61

B. Artículo Publicado: "Event-Driven Architecture for Intrusion Detection Systems Based on Patterns" **92**

Índice de figuras

1. Diseño de una red con un Sistema de Detección de Intrusiones basado en Red 2
2. Esquema general de un IDS 8
3. Arquitectura de Prelude 9
4. Arquitectura de Snort 10
5. Formato de una regla en Snort 11
6. Esquema de la base de datos OSVDB 14
7. Ejemplo de Grafo de Ataque sencillo 25
8. Representación de IDML en UML 28
9. Arquitectura del Sistema 30
10. Definición de un Evento de Interés normalizado. 36
11. Definición de alto nivel de un incidente. 41
12. Diagrama de secuencia de la arquitectura 46
13. Configuración del caso de uso. 47
14. Transacciones del Motor de Eventos. 49
15. Diagrama de Bloques del prototipo 51
16. Pantalla principal del prototipo 52
17. Ventana emergente de aviso de incidente 53
18. Ejemplo de Regla de Snort 59
19. Ejemplo de Regla con negación de dirección IP 59
20. Ejemplos de Rangos de Puertos 60
21. Ejemplo de negación de puerto 60
22. Ejemplo de operador bidireccional 61
23. Ejemplo de uso de referencias 63
24. Ejemplo de reglas Classtype 67
25. Ejemplo de prioridad de la regla 67
26. Mezcla de Bytecode Binario y Texto en un 'content' 68
27. Ejemplo de Negación 68
28. Regla de contenido con el modificador nocase 69
29. Regla de Content Combinado, Offset y Depth 70
30. Ejemplo de uso de la distancia 71
31. Ejemplo de uso de within 71
32. Uso de byte jump 76

Índice de listados de código

1. Ejemplo de vulnerabilidad según OSVDB 16
2. Ejemplo de vulnerabilidad según NVD 18
3. Esquema XML que sigue la entrada del módulo traductor 32
4. Entrada de un evento al modulo traductor 32
5. Esquema XML que sigue la salida del módulo traductor 35
6. Traducción de un evento a formato IDML 37
7. Incidente enviado a la Capa de Decisión 44

Abstract

En el campo de las Tecnologías de la Información la seguridad de los sistemas es uno de los factores más importantes a tener en cuenta. Sin una buena estrategia de seguridad, una de las partes más importantes y a la vez más vulnerables del sistema, los datos, podrían verse seriamente amenazados. Entre los componentes que ayudan a los administradores en las tareas de seguridad encontramos los Sistemas de Detección de Intrusiones o IDS.

Los Sistemas de Detección de Intrusiones son normalmente uno de los mecanismos básicos que se usan cuando se definen medidas de seguridad dentro de cualquier organización. Las soluciones actuales de IDS están únicamente enfocadas al descubrimiento de ataques simples, atómicos. Estos ataques individuales no tienen significado aparente por sí solos, convirtiéndose un conjunto de estos ataques individuales en un ataque mucho más complejo. Un ataque complejo sería el conjunto de pasos que un atacante debe llevar a cabo dentro de Sistema de Información Crítico para alcanzar el objetivo que busca. Sin embargo, existen pocas líneas de investigación activas que estén intentando corregir ese problema. Éste es el caso de la agregación y correlación de eventos cuando se trata con ataques complejos, o la mejora en la flexibilidad cuando se trata con diferentes versiones o mutaciones de un ataque concreto.

Con el fin de resolver estos problemas planteados, en este trabajo se presenta una arquitectura multicapa dirigida por eventos basada en el concepto de patrones. Además, a esta arquitectura vamos a añadirle cierta lógica de reacción con el objetivo de solucionar o mitigar los problemas causados por este tipo de ataques.

Las arquitecturas actuales de IDS están basadas en la búsqueda de patrones fijos. Estos patrones fijos obligan a que cuando un IDS busca un posible ataque éste sea cometido exactamente tal y como el IDS lo está esperando. Este hecho dificulta la vigilancia y la localización de variaciones de los ataques conocidos. Por ejemplo, el virus *Netsky* puede mutar en 36 formas diferentes. Esta situación puede provocar que el control de un sistema se haga inmanejable, puesto que con los sistemas actuales es necesario introducir cada una de las posibles variaciones de cada ataque para poder ser detectado.

La arquitectura presentada incluye conceptos como los grados de similitud y de credibilidad, presentados ambos como un enfoque probabilístico que nos ayude a hacer frente a las posibles variaciones de un ataque. Con la ayuda de esos dos valores probabilísticos, esta arquitectura soporta una mayor flexibilidad que nos permite conocer el nivel de veracidad que el sistema asigna a cualquier ataque detectado.

1. Introducción

Cotidianamente solemos referirnos al concepto de la seguridad como la ausencia de riesgo o también a la confianza en algo o alguien. Sin embargo, el término puede tomar diversos sentidos según el área o campo al que haga referencia. En el campo de la Informática, se entiende por *seguridad* aquel estado de cualquier sistema que nos indica que este está libre de riesgo o daño, es decir, un sistema se dice *seguro* cuando está libre de cualquier peligro ya sea exterior o proveniente desde el interior. Se entiende como peligro o riesgo todo aquello que pueda afectar su funcionamiento directo o los resultados que se obtienen del mismo. Para la mayoría de los expertos el concepto de seguridad en la informática es utópico porque no existe un sistema 100 % seguro, puesto que es bastante difícil y costoso abarcar todos los campos que debe cubrir la defensa de un sistema. Esta defensa debe tener en cuenta tanto ataques externos o internos al sistema, como ocurrencias físicas que pueden derivarse de cualquier tipo de catástrofe natural.

Para que un sistema se pueda definir como seguro debe tener estas cuatro características:

1. Integridad: La información sólo puede ser modificada por quien está autorizado.
2. Confidencialidad: La información sólo debe ser entendible o descifrada por personas autorizadas.
3. Disponibilidad: El sistema debe estar disponible siempre que se lo necesite.
4. No Repudio (No-Rechazo o Irrefutabilidad): Propiedad que evita que se pueda negar la autoría de una acción realizada en el sistema.

En estos momentos la seguridad informática es un tema de obligado dominio para cualquier usuario de Internet, para no permitir que su información de caracter personal sea accesible de manera incorrecta.

La *Detección de Intrusiones* es uno de los métodos utilizados para proporcionar seguridad a una red de ordenadores. Para comenzar, es importante conocer qué es la *Detección de Intrusiones*: es el proceso de monitorizar redes de ordenadores o sistemas en busca de violaciones de políticas de seguridad. Es decir, es el proceso de comprobar que un ordenador, una red o un sistema completo con varias subredes funciona correctamente desde el punto de vista de las políticas de seguridad definidas por la organización. La *Detección de Intrusiones* está compuesta por tres elementos funcionales básicos:

- Una fuente de información que proporciona eventos de red o de sistema.
- Un motor de análisis (o motor de detección) que busca evidencias de intrusiones.

- Un mecanismo de respuesta que actúa según los resultados del motor de análisis y de una lista de reglas de decisión (motor de decisión) para subsanar o mitigar los problemas que el ataque está produciendo en el sistema, o al menos evitar su propagación.

La detección de intrusiones surge como mecanismo de seguridad en las redes de ordenadores. Se necesita saber qué es lo que se hace en el sistema monitorizado, quién es la persona o máquina que realiza algo, cuándo y cómo. En la actualidad se ha convertido en un elemento importante para proteger los sistemas ubicados en un entorno de Tecnologías de la Información. En la figura 1 podemos ver una red con un Sistema de Detección de Intrusiones (IDS) dentro de su arquitectura. En esta figura se observa que el Sitema de Detección de Intrusiones vigila cualquier paquete que va desde o hacia la red privada.

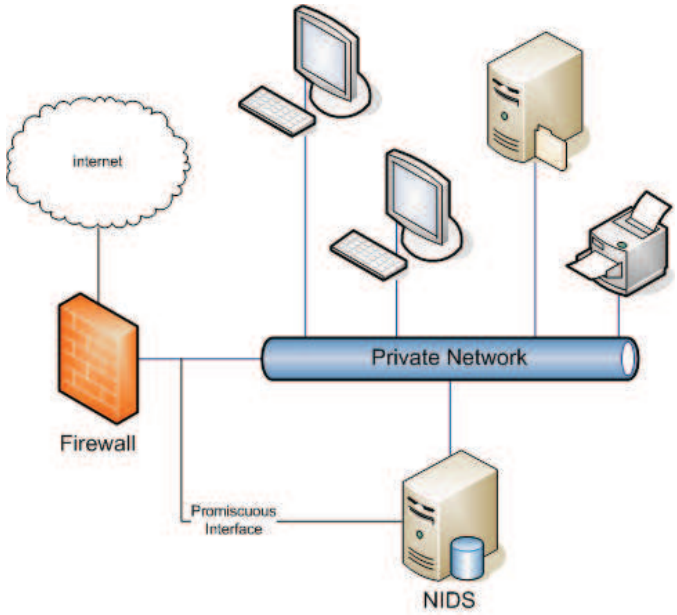

Figura 1: Diseño de una red con un Sistema de Detección de Intrusiones basado en Red

Existen multitud de Sistemas de Detección de Intrusiones, puesto que es un área en la que se investiga contínuamente desde mediados de la década de los 80. La mayoría de representaciones actuales están basadas en reglas, lo que provoca una falta de flexibilidad en los mismos. El enfoque propuesto en este trabajo está basado en eventos. Esto nos proporciona una granularidad mayor sobre cada ataque, puesto que cada conjunto de eventos que sean de interés nos proporcionará un ataque de mayor de complejidad.

En este trabajo se presenta un Sistema de Detección de Intrusiones basado en Eventos y estructurado en una arquitectura multicapa. Con esta arquitectura se soluciona el problema

sobre el tratamiento de los eventos que presentan los enfoques de los actuales IDS. La división en varias capas de la arquitectura otorga a la misma una independencia a la hora de poder modificar cualquiera de ellas, independientemente del resto. Esto además ayuda a eliminar complejidad en cada una de ellas, puesto que existe una división clara en cuanto al trabajo a realizar por cada una. En esta arquitectura se jerarquiza el contexto de manera que se simplifica su mantenimiento, es decir, en qué estado o fase se encuentra un ataque complejo. Esto era uno de los problemas por los cuales los IDS actuales no se basaban en eventos. Esta jerarquización del contexto ayuda a la implantación de una aproximación basada en eventos.

Con todo ello, indicar que el objetivo de este trabajo es el diseño de una arquitectura multicapa basada en eventos que permita ampliar las características de los IDS actuales y consiga detectar ataques complejos a partir de ataques atómicos. Esto se debe conseguir sin que suponga una sobrecarga excesiva del sistema en el que se introduzca esta arquitectura.

El trabajo aquí presentado se ha expuesto como parte de *The Second International Conference on Emerging Security Information, Systems and Technologies (SECURWARE)*, mediante el artículo *Event-Driven Architecture for Intrusion Detection Systems Based on Patterns*, el cual se puede encontrar en el anexo B. Este artículo ha sido expuesto en Agosto del 2008 en Cap Esterel, Francia.

El presente documento está dividido en distintas secciones. La primera sección muestra las herramientas utilizadas para desarrollar la arquitectura diseñada y el estado del arte de las soluciones actuales. En la segunda sección se muestra el diseño de la arquitectura propuesta, un ejemplo de funcionamiento y el diseño de un prototipo que pretende ejemplificar los conceptos presentados en este trabajo. En la tercera sección se muestran las conclusiones alcanzadas tras la realización de este trabajo y las vías futuras que permitirán continuar con el mismo.

2. Análisis

En esta sección se expondrán los objetivos que se pretenden alcanzar con este trabajo así como las herramientas estudiadas y las finalmente empleadas en la realización de este proyecto. Asimismo se incluye un estudio de las soluciones existentes.

El objetivo de este trabajo es sentar las bases de un Sistema de Detección de Intrusiones que sea capaz de detectar ataques complejos. Para ello se parte de lo que actualmente existe en la literatura y se propone una arquitectura con la que desarrollar el sistema. Como ataque complejo vamos a entender aquel ataque formado por una secuencia de eventos que pueden llegar en cualquier momento, en cualquier orden y desde cualquier dirección a la red que se esté monitorizando con el sistema. Como veremos, uno de los tipos de herramientas que necesitamos utilizar son los Sistemas de Detección de Intrusiones basados en red (NIDS, Network Intrusion Detection Systems).

En primer lugar se mostrarán los conceptos básicos de los Sistemas de Detección de Intrusiones (ver subsección 2.1), así como los IDS Open Source más representativos. En segundo lugar se mostrarán las bases de datos de vulnerabilidades Open Source más importantes (ver subsección 2.2). Estas bases de datos se han estudiado como alternativa a los IDS para propocionar información sobre los ataques que tienen registrados. En tercer lugar se explicará lo que es un evento y se mostrarán unos cuantos eventos que se pueden obtener a partir del Sistema de Detección de Intrusiones Snort (ver subsección 2.3), como muestra de cómo se forma un ataque complejo por medio de estos eventos. En cuarto lugar se explicará qué son las técnicas de correlación (ver subsección 2.4), mostrando además en profundidad una de dichas técnicas (los grafos de ataque). Estas técnicas serán utilizadas para agregar eventos, es decir para pasar de ataques simples a ataques más complejos, por medio de las relaciones encontradas entre esos ataques simples. Finalmente se verá un lenguaje basado en XML (ver subsección 2.5), IDML, el cual permite la definición de patrones de intrusión que se utilizarán para representar de forma estandarizada los diferentes flujos de información que tienen lugar en esta arquitectura.

2.1. Sistemas de Detección de Intrusiones

En este trabajo se mostrará una arquitectura para mejorar los Sistemas de Detección de Intrusiones actuales, por lo que un primer paso debe ser conocer qué son y cómo funcionan estos sistemas.

Entendemos como Detección de Intrusiones el proceso de monitorizar redes de ordenadores y sistemas en busca de violaciones de políticas de seguridad [9]. Está compuesta por tres elementos funcionales básicos:

- Una fuente de información que proporciona eventos de sistema.

- Un motor de análisis (o motor de detección) que busca evidencias de intrusiones.
- Un mecanismo de notificación que actúa según los resultados del motor de análisis y de una lista de decisiones (motor de decisión).

Las propiedades básicas que todo IDS debe cumplir son las siguientes:

- El IDS ha de ejecutarse continuamente sin que nadie esté obligado a supervisarlo; independientemente de que al detectar un problema se notifique a un operador, el funcionamiento habitual no debe implicar interacción con ningún operador.
- La aceptabilidad o grado de aceptación del IDS, es decir, los mecanismos de detección de intrusiones han de ser aceptables para las personas que trabajan habitualmente en el entorno.
- La adaptabilidad del mismo a cambios en el entorno de trabajo, es decir, que el IDS sea capaz de funcionar igual cuando se produce un cambio en la red que monitoriza.
- Todo IDS debe además presentar cierta tolerancia a fallos o capacidad de respuesta ante situaciones inesperadas; por ejemplo a un reinicio inesperado de varias máquinas o un intento de engaño hacia el IDS.

Además de estas propiedades, es importante que los IDS no proporcionen lo que se conoce como *falsos positivos*, es decir, no deben realizar demasiadas notificaciones de que ha ocurrido algo (una intrusión o anomalía, por ejemplo), cuando realmente no ha ocurrido nada peligroso para el sistema. Esto puede ocasionar que el administrador de la red o la persona encargada de la seguridad de la misma, no tenga demasiado en cuenta al IDS que le produce demasiados falsos positivos. Asimismo, deben producir el menor número posible de lo que se conoce como *falsos negativos*. Esto sucede cuando el sistema cree que todo va bien, y realmente está ocurriendo algo anormal. Esto es realmente peligroso que ocurra, ya que pueden estar produciéndose problemas y el encargado de la seguridad no recibiría notificación alguna de lo que está ocurriendo.

Qué son los Sistemas de Detección de Intrusiones y cómo surgen

La detección de intrusiones surge como un mecanismo de seguridad en las redes de ordenadores y en los propios ordenadores. Se necesita saber qué es lo que se hace en la red, quién es la persona o máquina que realiza algo, cuándo y cómo. Es una de las partes más importantes de estudio dentro del ámbito de la seguridad.

Antes de la aparición de la detección de intrusiones existían las auditorías de seguridad. La auditoría es el proceso de generar, almacenar y revisar eventos de un sistema cronológicamente. Dichas auditorías fueron incluidas por el Departamento de Defensa de los Estados Unidos como requisito en cualquier sistema de confianza. Esto está incluido en el llamado

"Libro Marrón" [13] de la serie de documentos publicados por dicho departamento relativos a los Sistemas de Confianza (denominados "Serie Arco Iris" debido al color de sus tapas).

Los objetivos de una auditoría son los siguientes:

- Permitir la revisión de patrones de acceso y el uso de mecanismos de protección del sistema.
- Permitir el descubrimiento de los intentos internos o externos de evitar los mecanismos de seguridad, así como permitir el bloqueo de dichos intentos.
- Permitir la detección de las posibles elevaciones de privilegios por parte de cualquier usuario.
- Y además, servir como garantía frente a los usuarios de que toda la información que se recoja sobre ataques e intrusiones será suficiente para controlar los posibles daños ocasionados en el sistema.

Estas auditorías de seguridad se realizaban de forma off-line, es decir, se realizaba un análisis después de que ocurriera el problema, con lo que ello conlleva. Está claro que con el aumento del tamaño de las redes de ordenadores, se hacía imposible analizar los eventos ocurridos por parte de una persona, con lo que surgía la necesidad de automatizar el proceso de análisis de dichos eventos. Además, hay que tener en cuenta que para una mayor seguridad, había que realizar un análisis on-line de la información. Así surgen a principios de los 80 los primeros Sistemas de Detección de Intrusiones, como el "Intrusion Detection Expert System" (IDES [6]) que definía un sistema de detección de intrusiones en tiempo real, el "Discovery" [26], que monitorizaba una base de datos en vez de un sistema operativo, o el "Haystack" [23] o el "Mutics Intrusion Detection and Alerting System" (MIDAS) [22], ambos pertenecientes a diferentes ramas del Departamento de Defensa estadounidense (al igual que IDES).

A partir de un estudio encargado por las Fuerzas Aéreas de EEUU surgió un sistema para dar solución al problema de los intrusos que se habían apoderado de cuentas de usuario. Dicho sistema era capaz de distinguir entre el comportamiento normal o inusual de las cuentas basándose para ello en patrones de uso, creados a partir del análisis estadístico de comportamiento de los usuarios. Esta idea fue posteriormente heredada en motores de detección mucho más modernos.

Conociendo un poco esta historia se obtienen las siguientes conclusiones. Por un lado, el uso importante de los análisis de auditoría (eliminando la información irrelevante o repetida de los informes de sucesos), y también la necesidad de distinguir entre comportamiento normal y anómalo. Por otro lado, la utilización de patrones de uso para realizar dicha distinción. Estas conclusiones son parte de los conceptos fundamentales que todavía se siguen en la creación y mejora de los IDS actuales.

Fuentes de Datos

Según el lugar en el que se realice la monitorización, es decir, según el lugar en el cual se recojan los datos para luego enviarlos al motor de análisis, los IDS se pueden dividir en:

- IDS basados en host (HIDS): se monitoriza un equipo, normalmente a nivel de sistema operativo. El propio IDS es vulnerable a ataques, ya que es parte del sistema. Ej: OSSEC [4].
- IDS basados en red (NIDS): analizan el tráfico de red inspeccionando los paquetes que circulan por ella, para lo cual se utilizan dispositivos de red con acceso a todos los paquetes que circulan por ella. Tienen como desventaja que no pueden analizar paquetes cifrados y ademas se pueden convertir en un cuello de botella en redes de alta velocidad. Ej: Snort [20].
- NIDS distribuidos (dNIDS): varios sistemas trabajan juntos en diferentes segmentos de la red, extrayendo información desde diferentes sensores y enviando dicha información a un sistema central. Son capaces de agregar eventos generados por diferentes fuentes, pudiendo así detectar mayor actividad maliciosa. Ej: Prelude [24].

Tipos de Análisis

La detección de intrusiones también se puede clasificar según los objetivos del motor de análisis, siendo los principales tipos la *detección de usos indebidos (misuse detection)* y la *detección de anomalías (anomaly detection)*.

Entendemos por detección de usos indebidos o detección de abusos aquel objetivo por el cual se intentan detectar usos indebidos o maliciosos en el sistema. Para ello se comparan patrones de ataque conocidos previamente (denominados firmas) con la información recogida previamente en busca de coincidencias. Tienen el problema de que no pueden detectar nada que no haya sido establecido al definir las firmas. Como consecuencia, estos sistemas son incapaces de detectar nuevos tipos de ataques que puedan surgir, teniendo que ser continuamente actualizada su base de datos de firmas.

En la *Detección de anomalías* se comparan los perfiles permitidos de todas las aplicaciones monitorizadaslos con el comportamiento del tráfico actual de la red. Todo lo que se salga de lo permitido se considera un ataque. Es un modelo bueno para detectar nuevos tipos de ataque, aunque surge el problema de que las anomalías podrían ser simplemente un nuevo comportamiento normal del sistema. Como consecuencia de un cambio en el comportamiento de los usuarios, este tipo de sistemas generan muchos falsos positivos.

La mayoría de los Sistemas de Detección de Intrusiones son de detección de usos indebidos, de anomalías o una mezcla de ambos. En la figura 2 se muestra un esquema general de un IDS. Para ver más claramente la diferencia entre los esquemas de detección de anomalías y de usos indebidos, imaginemos un sistema de detección basado en monitorizar las máquinas origen desde las que un usuario sospechoso se conecta a nuestro sistema. Si se tratara de un

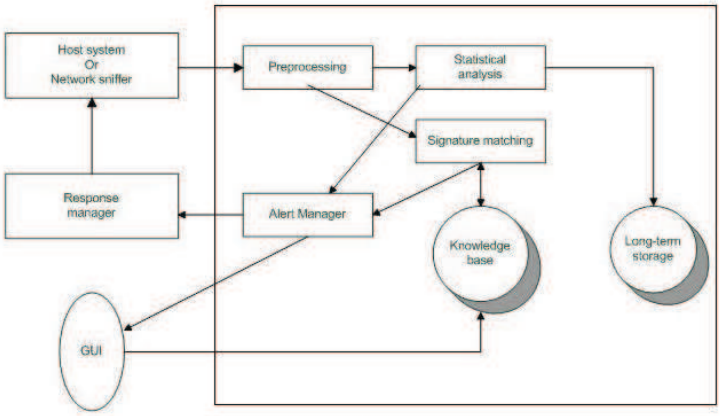

Figura 2: Esquema general de un IDS

modelo basado en la detección de anomalías, seguramente mantendrá una lista de las dos o tres direcciones más utilizadas por el usuario legítimo, alertando al responsable de seguridad en caso de que el usuario se conecte desde otro lugar. Por contra, si se tratara de un modelo basado en la detección de usos indebidos, mantendrá una lista mucho más amplia que la anterior, pero formada por las direcciones desde las que sabemos con una alta probabilidad que ese usuario no se va a conectar, de forma que si detectara un acceso desde una de esas máquinas, entonces es cuando el sistema tomará las acciones oportunas.

En las siguientes subsecciones se presentan los dos Sistemas de Detección de Intrusiones más utilizados en la actualidad.

2.1.1. Prelude

Prelude es un sistema detector de intrusiones híbrido (basado en máquina y en red) modular y distribuido. Debido a su comportamiento híbrido, Prelude es un producto que permite a todas las aplicaciones de seguridad disponibles presentar un informe a un sistema centralizado. Para lograr esto, Prelude se basa en IDMEF (Intrusion Detection Message Exchange Format) [3], estándar del IETF, que permite a diferentes tipos de sensores generar eventos utilizando un mismo lenguaje unificado. Prelude puede encontrar rastros de actividades maliciosas de multitud de sensores a fin de verificar mejor un ataque y a fin de llevar a cabo de forma automática la correlación entre los diversos eventos. Esta es una herramienta Open Source muy completa y comienza a tener cierta fama por su capacidad de funcionar de manera distribuida.

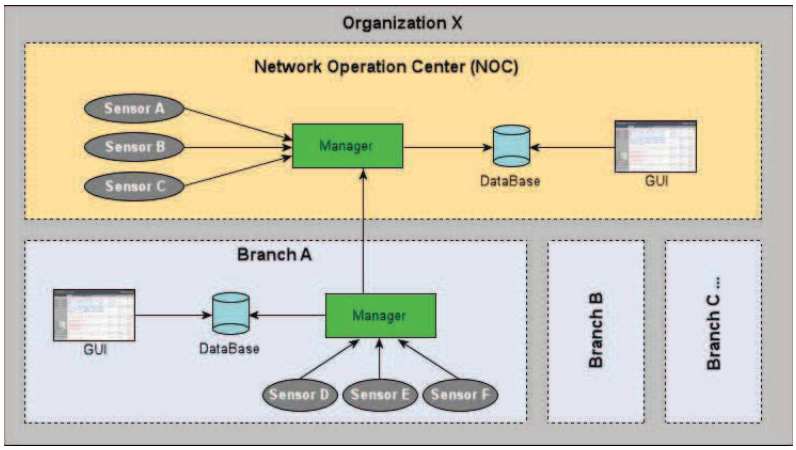

Figura 3: Arquitectura de Prelude

Se compone de tres partes bien definidas:

- Sensores. Itegran tanto NIDS como HIDS, siendo Prelude-nids su sensor NIDS por excelencia. Prelude-nids funciona con las reglas para la detección de alertas de otro Sistema de Detección de Intrusiones, Snort, el cuál veremos a continuación en la sección 2.1.2. Además integra un HIDS llamado Prelude-lml. Pero no sólo integra sensores propios, sino que puede integrar sensores externos. Para ello incorpora una librería libprelude que convierte cualquier programa en un sensor.

- Manager. Recibe, procesa y registra las alertas generadas por los sensores en una base de datos. La comunicación de los sensores con el manager se realiza mediante SSL (Security Socket Layer) [8].

- Frontend. Interfaz para visualizar las alertas generadas por la arquitectura.

Entre sus características encontramos que es modular, es híbrido y distribuido. Además la salida se puede generar en texto plano, en XML, haciendo un volcado en una base de datos o pasando la salida a otro manager. En la figura 3 podemos ver la arquitectura de este Sistema de Detección de Intrusiones.

2.1.2. Snort

Entre las herramientas estudiadas para la realización de este trabajo destaca Snort. En los siguientes párrafos se analizará a fondo esta herramienta.

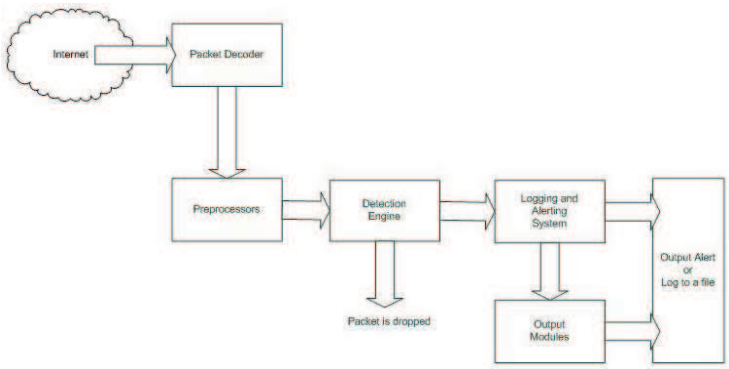

Figura 4: Arquitectura de Snort

Introducción a Snort

Snort [20] es un NIDS Open Source, rápido, flexible y que puede ser utilizado también como sniffer. Es un NIDS suficientemente probado, siendo actualmente el estándar de facto, es decir, el NIDS más reconocido y usado. Por defecto utiliza técnicas de detección de firmas y anomalías no estadísticas. Además es un software muy flexible que ofrece capacidades de almacenamiento de su log tanto en archivos de texto plano como en cualquier base de datos, como MySQL, Oracle o PostgreSQL.

Snort se puede utilizar de tres modos diferentes: Sniffer de red, registro de paquetes e IDS. En el modo de sniffer de red, Snort muestra el contenido que atraviesa la red en la que se encuentra instalado sin filtrar. Existen otras herramientas más potentes que Snort para actuar de sniffer, como *Tcpdump* o *Wireshark*. La única gran utilidad que tiene el modo sniffer en Snort es comprobar que éste funciona de forma correcta visualizando los paquetes de la red monitorizada. El modo de registro de paquetes es similar al modo sniffer, aunque nos permite registrar en disco los paquetes para que éstos puedan ser estudiados con posterioridad. Finalmente, el modo de funcionamiento más importante es el modo IDS. En este modo, Snort registra los paquetes sospechosos que atraviesan su red. Para trabajar en este modo, Snort utiliza unos ficheros de configuración que son los que le dicen a éste qué ficheros de firmas o reglas utilizar para decidir si un paquete se debe considerar sospechoso o no. Asimismo, Snort dispone de varios modos de alerta, dependiendo del nivel de detalle que queramos obtener de cada alerta obtenida.

Diferentes componentes de Snort

El motor de Snort está dividido en varios componentes:

- El decodificador de los paquetes. Este componente es el que se encarga de coger los paquetes de diferentes interfaces de red para prepararlos para ser preprocesados o envi-

ados directamente al motor de detección, es decir, las tramas capturadas se estructuran en un paquete bien formado.

- El preprocesador de dichos paquetes. Este es un componente o plugin que es usado por Snort para arreglar o modificar las tramas que le pasa el decodificador, antes de que el motor de detección haga su trabajo, es decir, prepara los datos para ser analizados contra reglas del motor de detección de Snort.
- El motor de detección, que realiza la comparación de los paquetes entrantes contra la base de datos de firmas. Es el responsable de detectar cualquier intrusión en el sistema.
- El logging y el sistema de alerta. Estos, dependiendo de lo que detecte el motor de detección, loguearán o generarán una alerta, siendo los logs almacenados como ficheros de texto en formato "tcpdump" u otro.

A continuación se explica un poco más en profundidad aquellas partes más importantes de Snort, como son su motor de detección y su base de datos de reglas (que en la clasificación anterior se encuentra dentro del propio motor de detección puesto que es éste el que hace uso de ella).

Base de Datos con las reglas de Snort

Es la base de datos que contiene información en forma de reglas. Es la que se encarga de suministrar información al motor de detección de Snort para que éste pueda realizar el mapeo entre lo que está llegando desde la red monitorizada y posibles ataques ya almacenados de antemano en formato de reglas. Este conjunto de reglas es extensible y es una de las características que hacen de Snort un NIDS tan extendido. Es posible obtener un buen conjunto de reglas desde la propia web de Snort, de forma gratuita. Además es sencillo añadir nuevas reglas y hacerlas públicas para que la comunidad internacional las pueda usar y chequear.

Estas reglas tienen un formato determinado, como podemos apreciar en la figura 5.

Figura 5: Formato de una regla en Snort

Los campos de la cabecera aparecen en todas las reglas de Snort. El resto son opcionales, y algunos de ellos pueden aparecer varias veces seguidas. Para ver una descripción completa sobre las reglas, su formato y sus campos véase el Anexo A. La cabecera de la regla contiene

la información que define el *¿quién?*,*¿dónde?* y *¿qué?* de un paquete, así como también qué hacer en caso de que sea detectado un paquete con todas las cualidades indicadas en la regla. El primer campo en una regla es la acción de la misma. Ésta le indica a Snort qué hacer cuando encuentra un paquete que concuerda con los criterios de la regla. Existen cinco acciones disponibles por defecto en Snort: alert, log, pass, activate y dynamic. El resto de campos indica el origen y el destino del paquete interceptado, indicando tanto la dirección IP como el puerto al que desean acceder y del que proviene el paquete.

Las opciones de cada regla forman parte del corazón de la máquina de detección de Snort, existiendo cuatro categorías principales de opciones de reglas:

- Meta-data. Proveen información acerca de la regla, pero no tienen ningún efecto durante la detección.
- Payload. Buscan datos en el interior de la carga útil del paquete y puede ser interrelacionados.
- Non-payload. Buscan datos en la otras partes del paquete diferentes a la carga útil.
- Post-detection. Estas opciones son disparadores de reglas específicas que ocurren después que una regla se haya disparado.

Todas las reglas están almacenadas en un conjunto de ficheros con extensión *.rules*, cuyo nombre indica de qué trata cada uno de ellos. Este conjunto de ficheros intenta cubrir los diferentes tipos de ataques o amenazas existentes, además de otras cosas que provean más información al sistema. Estos ficheros son totalmente modificables y extensibles, pudiéndose añadir nuevas reglas a las ya existentes y modificar las que ya hay. Además, es posible y recomendable crear nuestros propios ficheros de reglas, con lo que no mezclaríamos las que nos aporta Snort con las que nosotros añadamos. Estos nuevos ficheros sólo hay que darlos de alta en el fichero de configuración de Snort, llamado *snort.conf*, añadiendo en él una nueva línea con el nombre de nuestro nuevo fichero de reglas:

```
include $RULE_PATH/mis_reglas.rules
```

Motor de Detección de Snort

El motor de detección de Snort es la parte más importante de este NIDS. Es la parte del mismo encargada de supervisar los paquetes que llegan a la red monitorizada para encontrar posibles ataques o actividades anómalas dentro de esos paquetes. El motor de detección permite registrar, alertar y responder ante cualquier ataque previamente definido en su base de datos de reglas. Para este propósito utiliza el conjunto de reglas de la base de datos del propio Snort, mapeando todas estas reglas contra los paquetes que le van llegando desde el preprocesador. Si un paquete coincide con lo que una regla busca, entonces se realiza la

notificación al Motor de Decisión de Snort para que éste realice la acción oportuna que tiene que realizar para esa regla.

2.2. Bases de Datos de Vulnerabilidades

En esta subsección se mostrarán dos de las bases de datos de vulnerabilidades Open Source más importantes. Estas bases de datos nos proveen información acerca de las vulnerabilidades que se encuentran en los productos tanto software como hardware actuales. Son otro punto de información importante aparte de las propias bases de datos que aportan los IDS, definiendo cada una de ellas las vulnerabilidades en su propio formato, y poniendo dicha información al alcance de todo el mundo. En estas bases de datos se ha analizado la estructura y la información que poseen intentando obtener a partir de ellas todos los datos sobre vulnerabilidades que necesitemos. Estos datos servirían para ser capaces de generar de manera (semi-)automática reglas, las cuales deben permitir a un algoritmo de clasificación de tráfico conocer patrones que den lugar a ataques ya reconocidos por una comunidad de expertos.

Las Bases de Datos de Vulnerabilidades se encargan de mantener y administrar las vulnerabilidades de los productos informáticos conforme van apareciendo. Entre estas bases de datos algunas de las más importantes son OSVDB [18] y NVD [17].

2.2.1. OSVDB (Open Source Vulnerability Database)

OSVDB [18] es una base de datos de vulnerabilidades independiente y de código abierto, en la cual todo el mundo puede colaborar. OSVDB pretende que la comunidad internacional disponga de una base de datos de vulnerabilidades de libre distribución, que sea y esté actualizada con el esfuerzo de toda la comunidad, y que sea independiente de fabricantes y necesidades comerciales. Sus objetivos son proporcionar información técnica exacta, detallada, actualizada e imparcial para que los administradores de redes la puedan utilizar en su beneficio. En la figura 6 podemos observar el esquema de esta base de datos de vulnerabilidades.

Como base de datos que es propone una forma propia de almacenar los datos en ella describiendo las propias tablas, los campos que forman dichas tablas y las relaciones entre ellas. La información incluida por OSVDB en cada entrada de la base de datos de vulnerabilidades incluye: El asunto de la vulnerabilidad, el título de la misma, fechas y una corta descripción de la propia vulnerabilidad. No se muestra toda la información incluida en las entradas que se pueden visionar vía web, pero esa información sí que está almacenada en la base de datos que se puede descargar y puede ser publicada más tarde.

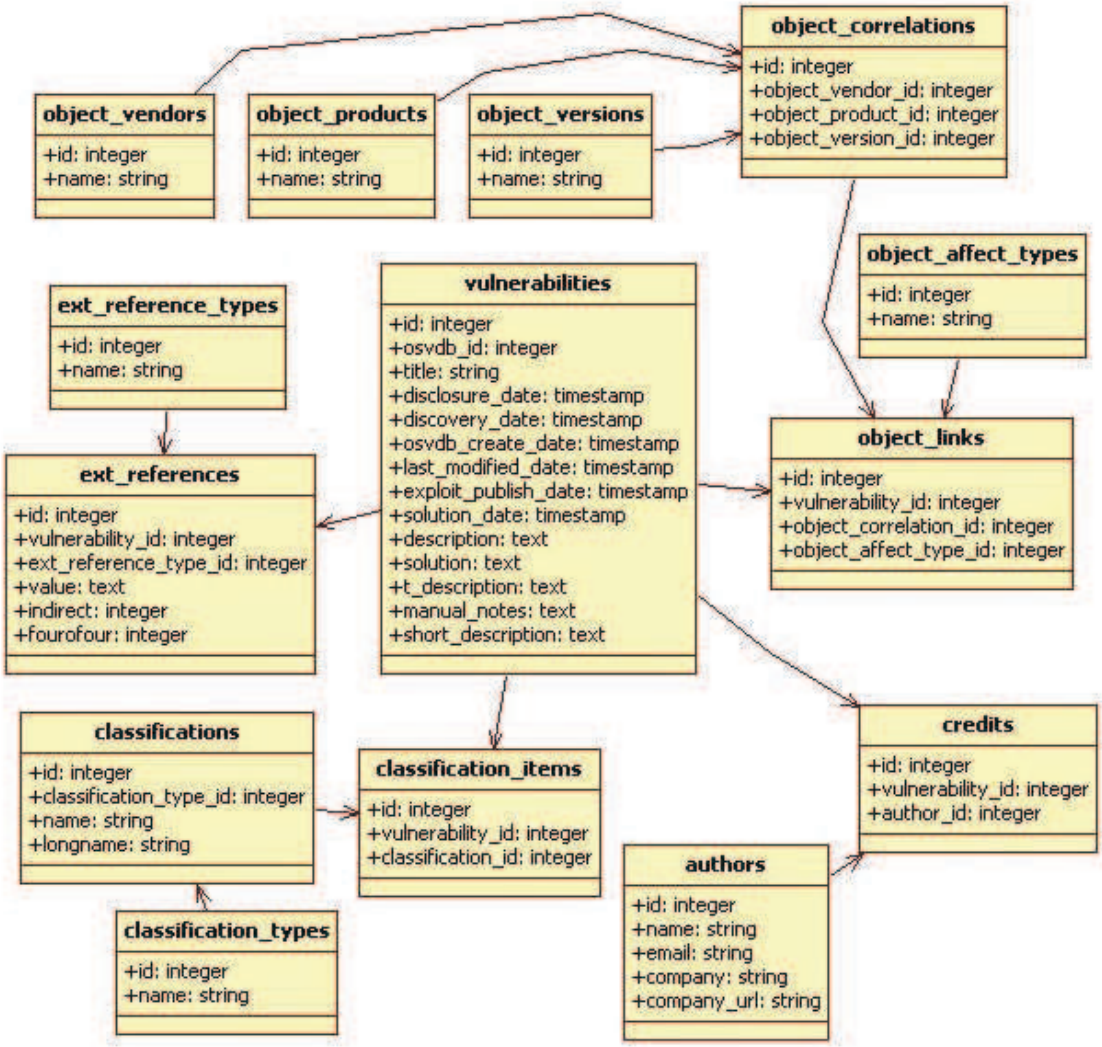

Figura 6: Esquema de la base de datos OSVDB

A continuación podemos ver una breve explicación de cada uno de los elementos que forman parte de la base de datos OSVDB:

- El título de la vulnerabilidad es un título conciso y único, que ayude a distinguir una vulnerabilidad de otra. En general, el título incluye el fabricante, el producto, el fichero, rutina o función que ocasiona la vulnerabilidad y el tipo de ataque. El título refleja el tipo de ataque o el privilegio que se consigue con él. Un ejemplo de título, que podemos observar en el listado 1, sería: *"B5Q Sitestats for Joomla rssfeeds.php baseDir Variable Remote File Inclusion"*.

- En cuanto a las fechas que se almacenan, se incluyen tres. La más importante es la fecha en la que se publica la vulnerabilidad, la cual se obtiene desde el aviso original o desde listas de correo. Las otras dos son las fechas en las que la vulnerabilidad fue descubierta (que es anterior a su publicación) y la última es la fecha en la que se publicó el código sobre explotación de la vulnerabilidad.

- En cuanto a la descripción que se incluye, lo que se intenta con ella es aportar toda la información posible de forma breve, intentando que ésta incluya la mejor información sobre la vulnerabilidad. Entre la información que debe incluir está el nombre del fabricante, las versiones afectadas, la naturaleza del problema y los privilegios que puede conseguir un atacante mediante la vulnerabilidad referida.

- Otros de los campos a incluir para describir la vulnerabilidad son la clasificación dada a la vulnerabilidad por OSVDB, la localización de la vulnerabilidad (acceso a ella de forma remota, directamente a través de la consola, de forma telefónica, desconocida, etc.) y el tipo de ataque (criptográfico, denegación de servicio, hijacking, manipulación de entradas, etc.).

- Uno de los campos importantes es el de impacto. En él se define el nivel de impacto que puede tener esa vulnerabilidad, lo cual es una información muy valiosa para los administradores de redes. Se definen tres niveles de impacto: Pérdida de confidencialidad, pérdida de integridad y pérdida de disponibilidad.

- La base de datos incluye un campo en el que se informa si la vulnerabilidad está disponible y publicada, si existe pero no está publicada o si simplemente es un rumor. Este campo sirve para describir mejor el problema, puesto que a veces no se encuentra disponible toda la información deseada.

- Finalmente se incluye un campo de uso interno que describe información específica sobre cómo se encuentra esa vulnerabilidad dentro de la propia base de datos; es decir, si se encuentra verificada, si contiene algún error, etc.

Como dato negativo para el objetivo de este trabajo está el hecho de que la forma de añadir nuevas vulnerabilidades a la base de datos es manual. Todas las personas que lo deseen se pueden convertir en contribuidores de este proyecto, para lo cual solamente se deben registrar. Una vez registrados, su tarea consistirá en rellenar los campos que conforman una vulnerabilidad (los campos corresponden a las tablas definidas) con los datos que puedan encontrar entre las referencias que se proporcionan. Los datos introducidos en esas tablas no tienen que cumplir un formato establecido, con lo cual la información que posee es más difícil de automatizar. Además, OSVDB tarda más tiempo en sacar las vulnerabilidades que otros lugares de Internet, como pueden ser listas de correo o sitios web, puesto que su forma de añadir las nuevas vulnerabilidades a la base de datos es, como ya hemos comentado, "manual".

A continuación se muestra un ejemplo de una vulnerabilidad, en la que se pueden observar los campos descritos en el párrafo anterior.

Listado 1: Ejemplo de vulnerabilidad según OSVDB

```
<vuln osvdb_id=''29287'' osvdb_create_date=''2006-09-29 11:21:47'' last_modified_date=''2007-05-12
    03:02:16''>
  <osvdb_title>B5Q Sitestats for Joomla rssfeeds.php baseDir Variable Remote File Inclusion<
      /osvdb_title>
  <disclosure_date>2006-09-29 11:18:52</disclosure_date>
  <discovery_date>2006-09-14 00:00:00</discovery_date>
  <exploit_publish_date>1970-01-01 00:00:00</exploit_publish_date>
  <location_remote>1</location_remote>
  <attack_type_input_manip>1</attack_type_input_manip>
  <impact_integrity>1</impact_integrity>
  <exploit_unknown>1</exploit_unknown>
  <vuln_web_check>1</vuln_web_check>
  <products>
    <product affected=''Affected''>
      <vendor_name>Joomla!</vendor_name>
      <base_name>BSQ Sitestats</base_name>
      <version_name>1.8.0</version_name>
    </product>
  </products>
  <ext_refs>
    <ext_ref type_name=''Secunia Advisory ID'' indirect=''0''>21859</ext_ref>
    <ext_ref type_name=''Other Advisory URL'' indirect=''0''>http://secunia.com/
        secunia_research/2006-63/</ext_ref>
    <ext_ref type_name=''Mail List Post'' indirect=''0''>http://archives.neohapsis.com
```

```
        /archives/bugtraq/2006-09/0492.html</ext_ref>
    <ext_ref type_name=''Vendor URL'' indirect=''0''>http://developer.joomla.org/sf/
        projects/bsq_sitestats</ext_ref>
    <ext_ref type_name=''Related OSVDB ID'' indirect=''0''>29284</ext_ref>
    <ext_ref type_name=''Related OSVDB ID'' indirect=''0''>29285</ext_ref>
    <ext_ref type_name=''Related OSVDB ID'' indirect=''0''>29286</ext_ref>
    <ext_ref type_name=''CVE ID'' indirect=''0''>2006-7124</ext_ref>
    <ext_ref type_name=''ISS X-Force ID'' indirect=''0''>29269</ext_ref>
    <ext_ref type_name=''Bugtraq ID'' indirect=''0''>20267</ext_ref>
</ext_refs>
<credits>
    <credit>
        <author_name>Sven Krewitt</author_name>
        <author_company>Secunia Research</author_company>
        <author_email>remove-vuln@secunia.com</author_email>
        <company_url>http://secunia.com/</company_url>
    </credit>
</credits>
<ext_txts>
    <ext_txt type_name=''Technical Description'' language=''English'' revision=''1''>
        <text>This vulnerability is only present when the register_globals PHP
            option is set to \&amp;\#39;on\&amp;\#39;. This has not been the
            default setting for PHP installs since version 4.2.0 (22-Apr-2002).</
            text>
    </ext_txt>
    <ext_txt type_name=''Solution Description'' language=''English'' revision=''1''>
        <text>Upgrade to version 2.2.1 or higher, as it has been reported to fix
            this vulnerability. An upgrade is required as there are no known
            workarounds.</text>
    </ext_txt>
    <ext_txt type_name=''Vulnerability Description'' language=''English'' revision=''1
        ''>
        <text>BSQ Sitestats for Joomla contains a flaw that may allow a remote
            attacker to execute arbitrary commands. The issue is due to rssfeeds.
            php not properly sanitizing user input supplied to the \&amp;\#39;
            baseDir\&amp;\#39; variable. This may allow an attacker to include a
            file from a remote host that contains arbitrary commands which will be
             executed by the vulnerable script.</text>
    </ext_txt>
    <ext_txt type_name=''Short Description'' language=''English'' revision=''1''>
        <text>B5Q Sitestats 1.8.0 for Joomla rssfeeds.php baseDir Variable Remote
```

```
                    File Inclusion</text>
                </ext_txt>
            </ext_txts>
            <scores></scores>
    </vuln>
```

En el listado 1 se observa la descripción de una de las vulnerabilidades incluidas en esta base de datos. En la siguiente base de datos estudiada se mostrará la misma vulnerabilidad con otro formato (ver listado 2).

2.2.2. NVD (National Vulnerability Database)

NVD [17] es un servicio del gobierno de los Estados Unidos que registra las vulnerabilidades documentadas de los productos informáticos desde 1988. Basada en la CVE (Common Vulnerabilities and Exposures), actualmente incorpora información de 15.000 vulnerabilidades, con una media de 400 nuevas vulnerabilidades cada mes. Es de acceso libre, permitiendo la realización de consultas en el lenguaje OVAL (Open Vulnerability Assessment Language) [2], que es un lenguaje estandarizado utilizado para almacenar las vulnerabilidades con un formato que sea común. La base da datos NVD puede utilizarse por usuarios tanto de software de código abierto como propietario. Es una base de datos importante, puesto que está mantenida y actualizada por el gobierno de los Estados Unidos, unos de los países del mundo que más dinero gasta en el campo de la seguridad informática.

Comparando esta base de datos con la anterior, OSVDB, observando la misma vulnerabilidad (listados 1 y 2) en ambos sitios podemos ver que ambas bases de datos siguen un DTD diferente para cada una de ellas y que es automatizable. Solamente hay que saber qué es lo que se quiere automatizar. Las base de datos NVD es más concisa, en el sentido de que no incorpora "créditos" (nombre, email y compañía del descubridor) y solamente almacena una descripción. OSVDB sí que incorpora "créditos", así como una descripción técnica (donde se encuentra la vulnerabilidad), una descripción de la solución, otra descripción más general de la vulnerabilidad y otra descripción mucho más corta. Además, ambas incorporan referencias a sitios web que publican estas vulnerabilidades.

El único problema que encontramos para utilizar cualquiera de estas dos bases de datos, o incluso ambas, es que se trata de bases de datos de vulnerabilidades pero para ser utilizadas por un administrador de redes o experto en seguridad. No están orientadas al entendimiento de sus campos por parte de una máquina, sino que más bien están construidas para ser utilizadas por humanos.

Listado 2: Ejemplo de vulnerabilidad según NVD

```
<entry type= ‘‘CVE’’ name=‘‘CVE-2006-7124’’ seq= ‘‘2006-7124’’ discovered=‘‘2006-09-14’’ published
    =‘‘2007-03-05’’ modified=‘‘2007-03-07’’ severity=‘‘High’’ CVSS_score=‘‘7.0’’ CVSS_vector=‘‘(AV:R/AC:L/
    Au:NR/C:P/I:P/A:P/B:N)’’)>
        <desc>
                <descript source=‘‘cve’’>PHP remote file inclusion vulnerability external/rssfeeds.php in
                    BSQ Sitestats (component Joomla) 1.8.0, and possibly other version before 2.2.1,
                    allows remote attackers to execute arbitrary PHP code via the baseDir parameter</
                    descript>
        </desc>
        <loss_types> <sec_prot other=‘‘1’’/>
        </loss_types>
        <vuln_types>
                <input/>
        </vuln_types>
        <range>
                <remote/>
        </range>
        <refs>
                <ref source=‘‘BUGTRAQ’’ url=‘‘http://www.securityfocus.com/archive/1/archive
                    /1/447356/100//0/threaded’’ patch=‘‘1’’>20060929 Secunia Research: Joomla BSQ
                    Sitestats Component MultipleVulnerabilities</ref>
                <ref source=‘‘’’ url=‘‘http://secunia.com/secunia_research/2006-63/advisory/’’adv=‘‘1’’
                    patch=‘‘1’’ />
                <ref source=‘‘’’ url=‘‘http://developer.joomla.org/sf/sfmain/do/viewProject/projects.
                    bsq_sitestats’’/>
                <ref source=‘‘BID’’ url=‘‘http://www.securityfocus.com/bid/20267’’ patch=‘‘1’’>20267</ref>
                <ref source=‘‘OSVDB’’ url=‘‘http://www.osvdb.org/29287’’ patch=‘‘1’’>29287</ref>
                <ref source=‘‘XF’’ url=‘‘http://xforce.iss.net/xforce/xfdb/29269’’ patch=‘‘1’’>bsq-
                    sitestats-rssfeeds-file-include(29269)</ref>
        </refs>
        <vuln_softs>
                <prod name=‘‘BSQ Sitestats’’ vendor=‘‘Joomla!’’>
                        <vers num=‘‘1.8.0’’/>
                </prod>
        </vuln_soft>
</entry>
```

2.3. Eventos

En esta subsección se explica lo que entendemos por evento en general, y en particular en el área de la seguridad informática. Los eventos nos permitirán aumentar la granularidad de los ataques recibidos en el sistema, siendo capaces con ello de detectar ataques complejos. Esto no es posible en los Sistemas de Detección de Intrusiones actuales. Para la detección de estos eventos se han utilizado como base las reglas contenidas en Snort.

Un evento es, por antonomasia, algo que ocurre. Es la ocurrencia de un conjunto de circunstancias de forma prevista o imprevista y que se compone de un conjunto de elementos que hacen que éste sea posible. En el campo de la seguridad se define un evento de seguridad como cualquier suceso que pueda comprometer la confidencialidad, integridad o disponibilidad de un sistema de información. Si el evento compromete algún parámetro de seguridad, entonces este evento pasa a ser un *incidente*.

Existen eventos de seguridad de varios tipos: Ataques a servicios específicos, anomalías de protocolo, denegación de servicio, tráfico anormal, etc. Los sistemas de detección de intrusiones realizan el control de estos tipos de eventos de seguridad. Cuando se detecta uno de dichos eventos, el Sistema de Detección de Intrusiones generalmente realiza las siguientes acciones principalmente:

- Registrar aquellos eventos que se consideren susceptibles de ser analizados y que pueden aportar información acerca de intentos de ataque sobre el sistema monitorizado.
- Asociar un nivel de criticidad al evento, siendo este nivel dependiente del riesgo que supone para el sistema.

2.3.1. Definición de eventos de ataques

Como se comentaba en el inicio de esta sección, entendemos por evento de seguridad cualquier suceso que pueda comprometer la confidencialidad, integridad o disponibilidad de un sistema de información.

Los eventos mostrados aquí se han recogido de la base de datos de reglas de Snort 2.1.2. Para ello se han estudiado los diferentes ficheros de reglas que contiene esta base de datos, en la cual, además, se encuentran los diferentes ataques divididos según su tipo.

Los eventos con los que se empieza a realizar este trabajo son los expuestos a continuación.

Directorio Comprometido

Este evento estudia si el contenido de un paquete de red intenta acceder a cualquier directorio de los considerados comprometidos, como por ejemplo */bin/**.

Orden Comprometida

El siguiente evento captura si en el contenido de un paquete de red se observa alguna orden peligrosa, por ejemplo *ps %20*, entonces el paquete está mostrando indicios de ser sospechoso.

Secuencia SQL Peligrosa

Si el paquete va dirigido a los servidores de SQL, van al puerto en el que están configurados estos servidores, y además dentro del contenido podemos encontrar una cadena parecida a ésta:

```
s|00|p|00|\_|00|p|00|a|00|s|00|s|00|w|00|o|00|r|00|d|00|
```

dicho paquete es considerado como evento peligroso, puesto que incluye una cadena de texto que puede ocasionar problemas en el servidor de SQL, ya que es posible que se esté intentando atacar a dichos servidores.

Puerto Troyano Conocido

Cualquier paquete que vaya a uno de los puertos de los troyanos más conocidos, es sospechoso de contener un ataque. Este tipo de evento tendrá una alta probabilidad de ser un ataque real, puesto que si se observa un paquete dirigido a uno de los puertos de troyanos conocidos, podemos asignar una gran veracidad al hecho de que nos intentan atacar.

Bombardeo de puerto

Para detectar los ataques tanto de denegación de servicio (DoS) como de denegación de servicio distribuido (DDoS) es necesario comprobar que una cantidad importante de paquetes se dirigen al mismo puerto en un intervalo de tiempo muy pequeño. Por ejemplo, el demonio *inet.d* (que, entre otras muchas cosas, atiende pedidos para POP3), por defecto, atiende unas 40 solicitudes por minuto. Superado este máximo, el demonio deshabilitará el servicio en cuestión al considerar que se puede estar provocando un ataque por DoS. También hay que tener en cuenta que los ataques de DoS van dirigidos siempre hacia la red interna o hacia servidores de la propia red interna.

De las reglas de Snort definidas en los ficheros *dos.rules* y *ddos.rules* no se puede extraer un patrón de ataque bien definido para detectar estos tipos de amenazas, aunque éstas se pueden caracterizar en un grafo de ataque.

Tipo ICMP Peligroso

Los paquetes ICMP tienen un fin muy particular, por lo que cualquier uso que no sea común podría ser considerado como un evento peligroso. Se debe mostrar especial atención a los paquetes con un tipo ICMP en desuso (4-source quench, 5-redirect message, etc.), o con tipo "unassigned" (tipos 1, 2 y 7). Incluso es conveniente vigilar los paquetes ICMP de tipo 11 (Time Exceeded in Transit) que salen de nuestra propia red.

Comando POP3 Sospechoso

Busca entre los paquetes que se dirigen a la red interna, a los puertoss 110 ó 995, aquellos con un contenido específico dentro de su carga útil, como puede ser: |01| en una posición dada ó |1603| seguido de |01| dos posiciones después, etc. También suelen buscar palabras clave dentro del contenido: `USER`, `CAPA`, `DELE`, `TOP`, `STAT`, `RSET`, `AUTH`. Para ello busca en un sitio específico de la carga útil del paquete.

2.4. Técnicas de Correlación

Puesto que en este trabajo vamos a utilizar como fuente de información mínima el concepto de evento, necesitamos técnicas que nos ayuden a manejar dichos eventos. Además, necesitamos herramientas que nos permitan buscar relaciones entre estos eventos simples, por lo que necesitamos saber si la unión de varios eventos simples es capaz de formar un evento mayor que represente un ataque complejo.

Las técnicas de correlación aparecen para paliar el problema que surge cuando los eventos recibidos por un sistema se hacen inmanejables para analizarlos uno a uno. A partir de este punto, es necesario procesar esta información para evitar que cada evento se estudie de forma individual y aislada, y dotarlo del mayor significado posible en el contexto del sistema que se esté monitorizando. Hay que hacer hincapié en la diferencia entre las técnicas de correlación de eventos (o datos en general) y las técnicas de agregación, puesto que las segundas se refieren en exclusiva al proceso de adquirir datos de mayor nivel.

El término *correlación de datos* significa asociar conjuntos de eventos detectados a través de diversos medios, aplicar conocimiento para determinar si están relacionados, y en caso afirmativo, de qué manera y en qué grado.

En esta subsección se tratará de exponer lo que son las técnicas de correlación, cuáles se utilizan en el campo de la seguridad informática y, finalmente, la técnica que se ha adoptado en este trabajo.

En principio, entendemos por correlación la capacidad de establecer patrones de relación entre ataques o comportamientos similares, mostrados desde distintas máquinas para así determinar diferentes aspectos como, por ejemplo, si el atacante es la misma persona, si se trata de un ataque coordinado, etc. A grandes rasgos, la correlación es una manera de diferenciar de forma eficaz incidentes de eventos. Es importante poder valorar si un evento corresponde con un incidente real o no. Si explicamos la correlación de otra forma, la entendemos como la búsqueda de puntos en común entre multitud de datos obtenidos de diferentes sensores de forma que se pueda recoger información de distintos segmentos de red, pudiendo estar relacionada esta información. Además, la correlación trata de ofrecer una visión global de la situación en la que se encuentra un sistema que recibe continuamente paquetes de datos desde el exterior y detectar posibles ataques no catalogados con anterioridad.

A medida que pasa el tiempo y que cada vez es más importante Internet, y por extensión, su seguridad, se confirma que la correlación de los datos obtenidos de las diversas fuentes de información para componer un nivel superior de abstracción, es uno de los elementos más importantes en el futuro de los IDS. Por medio de la correlación, los IDS deberán ser capaces de relacionar la información obtenida de gran parte de los elementos que conforman una infraestructura de red para lograr así obtener una visión global de las actividades que se están llevando a cabo dentro de la red monitorizada. La correlación inteligente de eventos de seguridad permite a los administradores de sistemas identificar diferentes tipos de amenazas y ataques, utilizando técnicas de correlación avanzadas que permiten a las corporaciones identificar patrones de intrusiones, eliminar la confusión y reducir alertas de falso positivo, mientras se identifican amenazas reales de seguridad para ayudar a acelerar el tiempo de respuesta.

El conocimiento necesario para realizar la correlación de eventos incluye tanto el conocimiento aportado por expertos como la información almacenada en bases de datos u otros repositorios.

Desde un punto de vista práctico, la correlación de datos es útil en varios aspectos, como son la validación de los datos de entrada a un sistema o la determinación del origen, magnitud o grado de amenaza de un incidente. La correlación de datos puede ayudarnos a realizar un estudio exhaustivo de un incidente y su impacto en el sistema. Este estudio nos puede ayudar en fases posteriores a identificar y responder de forma óptima a dicho incidente.

Las técnicas de correlación son un campo todavía muy abierto en el cual existen multitud de propuestas y taxonomías diferentes. Entre las propuestas más destacadas encontramos aquella que hace la siguiente división entre las distintas técnicas de correlación [7]:

- Técnicas basadas en el uso de plantillas. Las plantillas son patrones que se pueden detectar en el sistema. Ejemplo de plantillas serían un conjunto de direcciones IP en cierto orden o un patrón de ataque característico de un atacante en particular. Estas técnicas se implementan mediante sistemas de conocimiento basado en reglas o mediante el uso de redes neuronales. En estas técnicas, una o más condiciones identificables son la base para la correlación de eventos.

- Técnicas de correlación basadas en estadística. La correlación de eventos basada en la estadística trata de encontrar, a partir de métodos estadísticos, relaciones entre variables que se pueden usar para identificar patrones y eventos. Estas técnicas se basan en métricas, lo que hace que sean objetivas y aporten precisión y claridad cuando se analiza una posible intrusión. En estas técnicas, las variables representan, por ejemplo, el número de conexiones web de un host en otro, o el número de escaneos de vulnerabilidades realizadas por una máquina. Entre los métodos que implementan esta técnica están el índice de correlación de Pearson y la inferencia bayesiana. Estos métodos tienen el problema de que no descubren de forma completa la relación existente entre los diferentes eventos capturados por el sistema [14].

- Técnicas basadas en precondiciones y en consecuencias de ataques individuales [15, 25]. Estas técnicas correlacionan eventos si la precondición de una alerta satisface la consecuencia de alguna alerta anterior. Estas técnicas pueden descubrir la relación entre diferentes alertas y no están restringidas a escenarios de ataques conocidos. Una de las formas más utilizadas para implementar estas técnicas son los grafos de ataque [16], los cuales se estudiarán un poco más en profundidad en el siguiente apartado.

En cuanto a las métricas de correlación, según el estudio referenciado en [11] del Laboratorio Lincoln MIT, podemos encontrar las siguientes:

1. Priorización. Concierne a la habilidad de valorar las alertas, agrupar aquellas relacionadas y asignar prioridades según la probabilidad de que indiquen ataques.

2. Correlación de los pasos. Trata sobre la capacidad del sistema de correlación de reunir los pasos particulares de un ataque. Para ello toma como punto de partida la recopilación de las actividades del Sistema de Detección de Intrusiones, construyendo una imagen general de la cadena de acciones llevada a cabo por el atacante.

3. Correlación de varios sensores. Este proceso se lleva a cabo cogiendo información de al menos dos sensores diferentes para combinarla y mostrar una imagen más integrada de la actividad del atacante.

Además afirma que cada sistema de correlación produce los resultados con sus propios formatos, lo que dificulta combinar los resultados de forma automática.

En resumen, los objetivos finales de la correlación son los siguientes:

- Poder recoger información de diferentes sitios de una red para gestionar ataques distribuidos y/o coordinados.

- Distribuir la carga que supone la monitorización de grandes sistemas de red. De esta forma se pueden manejar los eventos de más "bajo nivel" por secciones dentro de una red. Gestionar todos esos eventos de manera centralizada, directamente ocasionaría un serio problema de cómputo.

Grafos de Ataque

Una de las técnicas de correlación más usadas es la basada en *Grafos de Ataque* [14, 19, 27]. Los grafos de ataque son una representación de todos los caminos a través de un sistema que acaban en un estado en el que un intruso o atacante ha conseguido de forma satisfactoria su objetivo (ver un ejemplo en la Figura 7). En los grafos de ataque, los nodos representan el estado en el que se encuentra un ataque y las aristas representan un determinado ataque o una transición entre estados de un ataque. Asignando probabilidades de éxito a estas aristas o costos representando el nivel de esfuerzo requerido por parte del atacante para cometer un

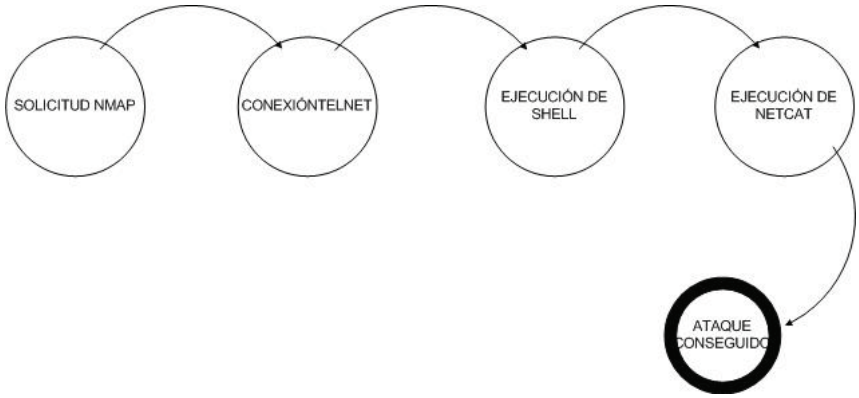

Figura 7: Ejemplo de Grafo de Ataque sencillo

ataque, varios algoritmos sobre grafos, como "camino más corto" o cualquier otro, pueden identificar los caminos o pasos que debe seguir un atacante para lograr cometer un ataque con mayor probabilidad de éxito.

Éste es un método flexible en el sentido de que permite definir de la misma forma multitud de ataques complejos. Además, permite tanto el análisis de los ataques provenientes de redes exteriores como de redes interiores. Pueden analizar el riesgo de un elemento específico de una red o examinar todas la posibles consecuencias provocadas por un ataque que ha tenido éxito. El inconveniente que tienen los grafos de ataque es que cada uno de ellos define de una forma fija la serie de pasos que un atacante debería seguir para realizar una acción peligrosa contra un sistema, lo cual hace que cuando un ataque varíe uno de los pasos el grafo no funcione como cabría esperar.

A la hora de utilizar grafos de ataque para analizar sistemas, este análisis necesita como entrada una base de datos en la cuál se puedan encontrar una buena variedad de ataques comunes, divididos en pasos lo más atómicos posibles. Además de estos ataques, es recomendable poder encontrar en esta base de datos valores como la configuración propia de la red que se está monitorizando, información topológica sobre dicha red y un posible perfil de atacante, con el cual se puedan conocer las características básicas que pueden distinguir a un posible atacante de un usuario normal (por ejemplo, uso de nombres de usuario normalmente restringidos a administradores). Con la información de los ataques se realiza un "mapeo" entre la información sobre la red y el perfil del atacante para crear un superconjunto de grafos de ataque.

Existen varios enfoques a la hora de representar ataques mediante grafos. En el enfoque asumido por este trabajo, cada nodo representa el estado en el que se encuentra un posible ataque. Un nodo será normalmente una combinación de máquinas físicas, niveles de acceso de usuarios y de efectos de posibles ataques, tales como troyanos o modificaciones del control

de acceso. Las aristas representarán el cambio de estado causado por la realización por parte del atacante de una acción simple, incluso si el atacante solamente ha conseguido acceso al sistema como un usuario normal del mismo.

Para generar el grafo de ataque se necesitan tres tipos de entradas. Plantillas de ataques, un fichero de configuración y un perfil de posible atacante. La plantilla de ataques representa ataques genéricos, conocidos o supuestos, incluyendo condiciones como la versión del sistema operativo sobre el que se tiene que ejecutar ese ataque para que sea posible. El fichero de configuración aporta información detallada acerca del sistema específico que se está analizando, incluyendo la topología de la red y la configuración de cada elemento particular de la red, tales como estaciones de trabajo, impresoras o routers. El perfil del atacante contiene información sobre las capacidades que se asumen sobre éstos, como puede ser la posesión y utilización de herramientas automáticas o *sniffers*. Aunque estas plantillas son una representación de métodos de transición de un estado a otro de amenazas conocidas o supuestas, sus combinaciones pueden proveer al sistema de descripciones de nuevas acciones ofensivas. Esto es, algún camino en el grafo que representa un ataque podría ser cubierto o recorrido por más de una amenaza conocida diferente. Con todo esto, los expertos en seguridad del sistema monitorizado son los responsables de analizar los posibles riesgos potenciales del propio sistema para construir todos los grafos necesarios para defenderlo [10].

Cada arista o transición de un grafo de ataque tiene un peso, el cuál representa la probabilidad de éxito o el tiempo medio para suceder. Las transiciones etiquetadas con un peso de 0 son, por lo general, omitidas.

Un camino corto o pequeño en el grafo de ataque representa un ataque de un coste pequeño. Si los pesos de las aristas son razonablemente válidos o aproximados a la realidad, el conjunto de todos los caminos óptimos, o casi óptimos, representará a las partes más vulnerables de la red.

2.5. Intrusion Detection Markup Language (IDML)

Para poder comunicar las diferentes capas de la arquitectura diseñada en este trabajo es necesario que los mensajes que fluyen de una capa a otra tengan un formato en común. Para lograr este proprósito se ha utilizado IDML.

IDML [12, 21] es un lenguaje XML que puede ser utilizado para la definición de patrones de intrusión. IDML posee una estructura que capacita a los expertos en seguridad a representar intrusiones de un modo sencillo. Define una serie de DTDs que permiten definir desde una propiedad a un patrón de intrusión completo:

- DTD *Property*. Consta de un nombre de atributo, un valor y adicionalmente una descripción de lo que representa dicha propiedad), que sirve para monitorizar el tráfico que tiene como destino cierta máquina.

- DTD *Event*. Representa un conjunto de propiedades, con campos como nombre, tiempo, duración y descripción. Sirve para agrupar varias propiedades añadiéndoles la fecha y hora en la que se produjo el evento y la duración de éste, lo que nos ayuda a representar ataques compuestos.

- DTD *EventCmpr* o Evento Comparador. Se utiliza para describir las condiciones de acierto de un patrón de intrusión.

- DTD *StateName*. Representa un estado, mediante su nombre, el o los enlaces con otros estados (que incluyen el *EventCmpr* y el siguiente estado), la acción a realizar y la descripción de dicho estado.

- DTD *IntrusionPattern*. Representa un patrón de intrusión completo. Incluye el nombre del patrón, su estado inicial y el tiempo de vida que puede estar vigente sin finalizar.

La relación entre estos DTDs se puede observar en la figura 8. Con todo esto es posible describir de una forma rápida y sencilla un patrón de intrusión de una manera estándar.

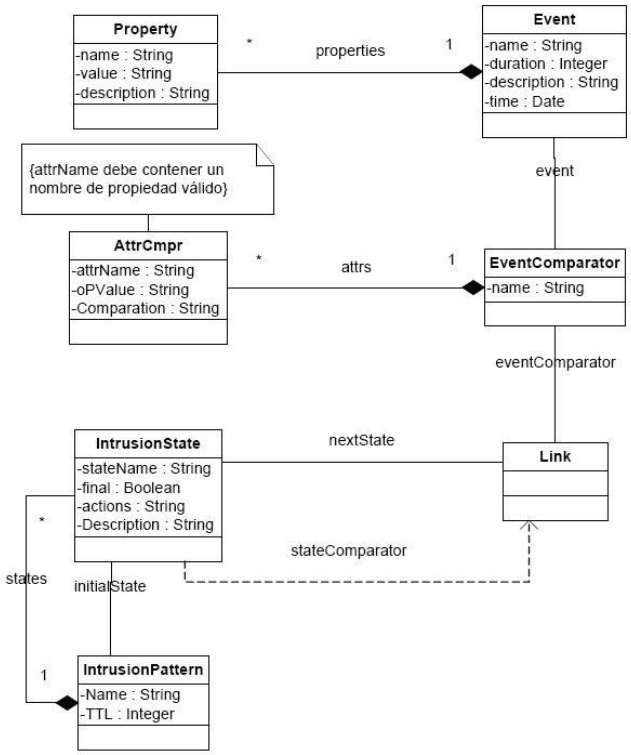

Figura 8: Representación de IDML en UML

3. Diseño y resolución del trabajo realizado

En esta sección se estudiará el diseño de la arquitectura propuesta en este trabajo. Esta arquitectura viene a solucionar el problema del manejo de ataques complejos y de la flexibilidad a la hora de tratar con variaciones de los mismos por parte de los Sistemas de Detección de Intrusiones actuales.

Se estudiarán los conceptos asociados a esta arquitectura para su entendimiento, entre los que encontramos algunos el flujo de datos que recorre toda la arquitectura desde que un paquete de red transmitido por la red monitorizada hasta que se decide qué hacer con dicho flujo de datos. También se presentarán otros conceptos relacionados con los matices aportados por el nivel de creencia que se posee sobre un posible ataque. Finalmente, la sección acaba con la presentación de un pequeño ejemplo como prueba de concepto que sirva para entender la arquitectura propuesta.

3.1. Arquitectura del Sistema

En esta subsección se explicarán los componentes de nuestro sistema, incluyendo las definiciones de las diferentes partes en las que se descompone nuestra arquitectura y las definiciones de los eventos más característicos con los que empezar a trabajar.

La propuesta desarrollada en este proyecto presenta tres capas bien definidas, como podemos observar en la figura 9. Estas capas están estructuradas de una forma modular, permitiéndonos realizar cambios en cualquiera de ellas sin afectar a las otras. Cada una de las capas está descrita en las siguientes subsecciones.

A modo de resumen general de la arquitectura presentada en este trabajo, vemos que cada una de estas capas tiene una función diferente dentro del marco general propuesto. La primera de estas capas, tal y como podemos ver en la figura 9, es la *Capa de Preprocesamiento (Preprocessing Layer)*. Esta capa se ocupa de examinar aquellos paquetes que se consideren de interés en tiempo de ejecución. Estos paquetes de interés son aquellos que pueden ocasionar posteriormente algún tipo de problema al sistema monitorizado, como pueden ser acciones que implican un riesgo importante, como el intento de conexión en una ventana de tiempo a distintos puertos de un mismo ordenador (escaneo de puertos). Cuando se encuentra un paquete relevante, esta capa informará a la siguiente mediante el envío de un evento. En la arquitectura propuesta, el preprocesador es equivalente a un NIDS completo que realiza un procesamiento basado en reglas.

La siguiente capa que vemos en la figura 9 es la *Capa de Detección de Eventos (Event Detection Layer)*. Esta capa, aprovechando los eventos que se reciben desde el preprocesador, aporta una visión de más alto nivel sobre lo que está ocurriendo. Consultando la información disponible en forma de patrones, determina si los diversos eventos recibidos constituyen un ataque complejo.

Por último, encontramos la *Capa de Decisión (Decision Layer)*. Esta capa ofrece una respuesta ante una secuencia de eventos previamente correlacionados por la *Capa de Detección de Eventos*. Cuando se determina la existencia de un ataque complejo la *Capa de Decisión* puede tomar decisiones acerca de la acción a realizar, bien directamente o bien informando al administrador de la red.

3.1.1. Capa de Preprocesamiento

La primera de las capas que conforman nuestra arquitectura, la *Capa de Preprocesamiento*, es la encargada de examinar en tiempo real los paquetes de interés que se transmiten a través de la red monitorizada. Esta capa determina si alguno de los paquetes se considera relevante en base a la información almacenada en una base de datos con reglas.

Esta capa está compuesta por un *Sistema de Detección de Intrusiones en Red* (NIDS) [1, 5], el cual realiza el preprocesamiento de los paquetes de la red basándose para ello en

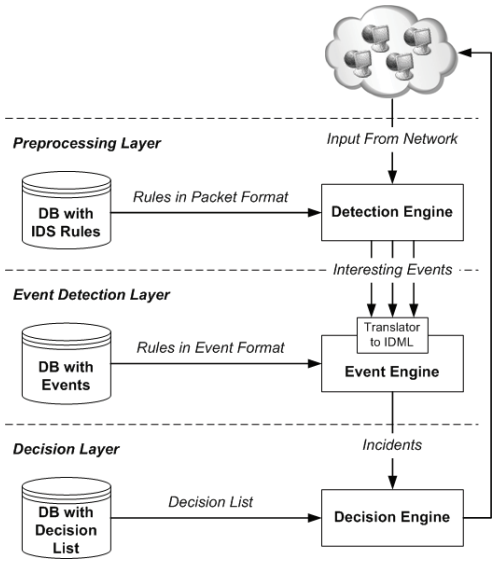

Figura 9: Arquitectura del Sistema

un conjunto de reglas predeterminadas. Cuando encuentra un paquete relevante, envía este evento de interés a la siguiente capa del sistema indicándole con ello que algo ha ocurrido en el primer nivel.

Por lo tanto, la *Capa de Preprocesamiento* es una caja negra en nuestro sistema, puesto que no son relevantes los detalles de cómo realiza su trabajo o los pasos que sigue para descubrir si un paquete se debe considerar sospechoso o no, sino que lo único que nos interesa de esta capa es su salida, lo cual permitiría en este sistema usar diferentes NIDS. Es en este punto donde hemos decidido optar por utilizar Snort, al ser éste el estándar de facto en el campo de los NIDS (ver sección 2.1.2). Esto asegura que utilizamos un NIDS actualizado día a día, cuya facilidad para incorporar nuevas reglas permite que esté continuamente actualizado y puesto a disposición de todos los usuarios.

Entre los elementos que forman esta capa, y que podemos ver en la figura 9, encontramos la *Base de datos con las reglas de Snort (DB with IDS Rules)* y el *Motor de Detección (Detection Engine)*, que en nuestro caso pertenece al NIDS Snort. La base de datos es la que contiene la información en forma de reglas. Sirve para aportar esas reglas al *Motor de Detección* para que éste pueda chequear si un paquete es peligroso o no.

Para hacer esta arquitectura todo lo distribuida como sea posible, y poder monitorizar grandes sistemas, esta capa podría extenderse para manejar diferentes redes. Cada uno de los NIDS colocados en esas redes tendrían que reportar todos los datos que captaran al mismo *Motor de Eventos* (ver sección 3.1.2). Incluso sería posible añadir diferentes NIDS en las

alert tcp $EXTERNAL_NET any –> $HTTP_SERVERS $HTTP_PORTS (msg: “WEB-ATTACKS /bin/ps command attempt”; flow:to_server,established; uricontent:“/bin/ps”;nocase; classtype:web-application-attack; sid:1328; rev:6; *similarity:60*)
alert tcp $EXTERNAL_NET any –> $HTTP_SERVERS $HTTP_PORTS (msg: “WEB-ATTACKS ps command attempt”; flow:to_server,established; uricontent:“ps %20”;nocase; classtype:web-application-attack; sid:1329; rev:6; *similarity:90*)
alert tcp $EXTERNAL_NET any –> $HTTP_SERVERS $HTTP_PORTS (msg: “WEB-ATTACKS wget command attempt”; flow:to_server,established; content:“wget %20”;nocase; classtype:web-application-attack; reference:bugtraq,10361; sid:1330; rev:6; *similarity:80*)
alert tcp $EXTERNAL_NET any –> $HTTP_SERVERS $HTTP_PORTS (msg: “WEB-ATTACKS uname -a command attempt”; flow:to_server,established; content:“uname %20-a”;nocase; classtype:web-application-attack; sid:1331; rev:5; *similarity:98*)

Tabla 1: Nuevas Reglas definidas para Snort con el concepto de Similitud asociado.

redes para disponer de más información y hacer que ésta sea más creíble, aumentando así la fiabilidad del sistema.

A continuación se explicarán los dos componentes que constituyen la *Capa de Preprocesamiento*.

Base de datos con las reglas de Snort

Es la base de datos propia del NIDS que se esté utilizando en la *Capa de Preprocesamiento*, en nuestro caso Snort (ver sección 2.1.2). Es la que se encarga de suministrar información a la *Capa de Preprocesamiento*, en la cual se encuentra el *Motor de Detección*.

Esta base de datos ha de modificarse ligeramente para añadir un valor numérico, el *Grado de Similitud prefijado* que indique cómo de similar debe ser el contenido o paquete encontrado por Snort para considerar que es el mismo que el que se muestra en la regla (a la que acompaña dicho grado). Este valor es el grado de similitud del paquete que está monitorizando nuestro NIDS con una regla. Está prefijado de antemano y añadido a cada regla o a cada conjunto de reglas semejantes. Para añadir este valor de similitud existen varias posibilidades. Por un lado podemos reutilizar el campo *priority* de las reglas de Snort, el cual indica la prioridad de la regla. Por otro lado, podemos utilizar un nuevo campo, llamado *similarity*, que incluimos para cada una de las reglas de Snort y que podemos añadir al final de las mismas. Podemos ver un ejemplo en la tabla 1.

Motor de Detección

Dentro de esta arquitectura, es el NIDS que se utiliza en la *Capa de Preprocesamiento*. Podemos hablar de él como un motor de detección basado en reglas; es decir, dentro de nuestro *Motor de Detección* habrán diferentes partes que se dedicarán a recoger el tráfico que viene desde el exterior (el tráfico en crudo), y a buscar determinados eventos en dicha información.

En el *Motor de Detección* se realiza un mapeo entre el conjunto de reglas que se encuentran en la *Base de Datos con las reglas de Snort* y los paquetes que están llegando continuamente desde la red monitorizada por el sistema. Cuando encuentre cualquier suceso de los supuestos interesantes, el *Motor de Detección* enviará a la siguiente capa del sistema (la *Capa de Detección de Eventos*) el tipo de suceso detectado junto con información de contexto y un margen de confianza (llamado *Grado de Similitud*, ver siguiente subsección), que puede ser menor, igual o mayor que el grado de similitud prefijado. Sólo cuando el grado de similitud calculado sea mayor o igual al prefijado, este suceso será reportado al nivel superior (capa de eventos). En esta parte del sistema encontramos información de estado dentro del propio contexto.

A modo de resumen tenemos que en la arquitectura propuesta encontramos el *Motor de Detección*, el cual se dedicará a recoger el tráfico que viene desde el exterior (el tráfico en crudo) para buscar determinada información coincidente con la almacenada en la base de datos. El objetivo será el de generar un suceso para el nivel superior (la *Capa de Detección de Eventos*). Cuando la información recogida en tiempo de ejecución se estime relevante, se generará este evento de interés, el cual será transmitido hacia la *Capa de Detección de Eventos*. En este punto de la arquitectura el flujo de datos estará compuesto por un documento en formato XML. Dicho documento incluirá, por un lado la información detectada por el NIDS utilizado en esta capa y en su propio formato original (en el caso de Snort será el fichero de log en formato texto); y por otro lado, el *Grado de Similitud* calculado para el suceso que se está enviando. Esto lo podemos ver tanto en el listado 3 que representa el DTD del documento que se envía de una capa a otra, como en el listado 4, que representa un ejemplo de la aplicación del mismo.

Listado 3: Esquema XML que sigue la entrada del módulo traductor

```
<xml version=''1.0'' enconding=''Big5'' ?>
<!ELEMENT interestingEventNIDS (nids:Event, similarity)>
        <!ELEMENT nids:Event (Log)>
                <!ELEMENT Log (#PCDATA)>
        <!ELEMENT similarity (percent)>
                <!ELEMENT percent (#PCDATA)>
```

Listado 4: Entrada de un evento al modulo traductor

```
<xml version=''1.0'' enconding=''Big5'' ?>
<interestingEventNIDS>
        <nids:Event>
                <Log>
                        [**] [1:9000003:1] Command execution attempted [**]
                        [Classification: A suspicious command was detected] [Priority: 2]
                        02/28-13:34:32.762408 155.54.205.251:40699 -> 155.54.205.80:23
                        TCP TTL:240 TOS:0x10 ID:0 IpLen:20 DgmLen:213
                        ***AP*** Seq: 0x8CFE1E61  Ack: 0xBF7138EC  Win: 0x1700  TcpLen: 20
                </Log>
        </nids:Event>
        <similarity>
                <percent>100</percent>
        </similarity>
</interestingEventNIDS>
```

Grado de Similitud

Uno de los principales problemas de las arquitecturas de detección de intrusiones basadas en reglas es la poca o nula flexibilidad que ofrecen a la hora de manejar dichas reglas. Estas arquitecturas esperan solamente paquetes de red que estén bien definidos, es decir, saben exactamente de antemano lo que están esperando recibir. Esto supone que para marcar un paquete como peligroso éste debería ser exactamente igual que alguno de los ya almacenados en la base de datos del IDS que se utilice en la red monitorizada. Si un paquete de red de los que el IDS acaba de recibir no es exactamente igual a lo que el sistema tiene almacenado de antemano, aunque sea muy parecido a algo ya existente, las arquitecturas actuales basadas en patrones no detectan este paquete como peligroso para su sistema. Por ejemplo, en el campo de los virus es muy normal que éstos muten, y no solo una vez sino varias veces. Supongamos, por ejemplo, una de las muchas posibles mutaciones que el famoso gusano *Netsky* puede presentar. Los sistemas actuales detectarían el gusano Netsky, tal vez sin ningún problema, pero si éste muta, el IDS sería incapaz de informar sobre dicha transformación del virus.

Como consecuencia de la falta de flexibilidad sobre los patrones esperados, las arquitecturas actuales son propensas a producir numerosos falsos negativos, puesto que no son capaces de reconocer ataques derivados de otros ataques ya existentes. Una de las vías que existen para detectar todas esas posibles mutaciones de ataques reales, como si fueran patrones realmente esperados, es definiendo todas las posibles variaciones que el ataque pueda presentar. Es decir, definir cada una de las posibles mutaciones o variaciones de los ataques

conocidos, con todo el esfuerzo que ésto supone. Obviamente, la introducción de todas las mutaciones de cada uno de los patrones puede considerarse una tarea imposible de manejar por un administrador.

En la arquitectura propuesta en este trabajo intentamos evitar esta falta de flexibilidad existente en las arquitecturas actuales. Para ello realizamos una asignación de valores probabilísticos a todos los paquetes que se introduzcan en el sistema y que sean marcados como sospechosos por el NIDS que estemos utilizando en la *Capa de Preprocesamiento*. En éste caso, asignamos dichos valores de probabilidad a todos los paquetes marcados como sospechosos por el NIDS Snort. Este valor de probabilidad, al que llamamos *Grado de Similitud*, representa cuánto de parecido debe ser un paquete de datos de los revisados por el NIDS para ser considerado igual que una regla de las almacenadas en su base de datos. Esto es, el valor de probabilidad es un valor que define el parecido entre lo que estamos recibiendo de la red (el paquete marcado como sospechoso por el NIDS Snort) y lo que tenemos almacenado como interesante o peligroso (las reglas que Snort almacena en su base de datos).

El *Grado de Similitud* calculado para un paquete sospechoso puede variar en un rango entre el 0 % y el 100 %, correspondiendo el valor del 0 % a la confianza total en que el paquete que acabamos de recibir no es finalmente sospechoso. Un valor del 100 % representa la certeza absoluta de que la información es relevante. Siguiendo con el ejemplo de *Netsky*, el gusano original tendrá una certeza del 100 %. Los IDS actuales solamente son capaces de detectar paquetes cuyo grado de similitud (para nosotros) es el 100 %; es decir, la definición del ataque original, por lo que cualquier mutación de este gusano, que en nuestra arquitectura tendrían un Grado de Similitud cercano al 100 %, no sería detectado por los sistemas actuales. Este valor podría ser calculado por el propio NIDS de la *Capa de Preprocesamiento*, Snort, aunque habría que modificar el código fuente de este para poder introducir esta nueva característica, o realizar un plugin que realizara esta tarea. Si esto no fuera posible, Snort no será capaz de detectar ninguna mutación de los ataques, y en consecuencia, solo podrá identificar ataques con un *Grado de Similitud* igual al 100 %. Para evitar en la medida de lo posible los falsos negativos, en el trabajo realizado vamos a descartar todos los eventos detectados cuyo *Grado de Similitud* esté por debajo de un cierto valor umbral. Para esto, cada regla definida en la *Base de datos con las reglas de Snort* mantendrá un valor umbral por debajo del cuál el evento será omitido, y por lo tanto no se considerará peligroso finalmente.

Para poder añadir realmente este *Grado de Similitud* a la arquitectura nos serviremos del lenguaje XML. Con ello, la *Capa de Preprocesamiento* enviará a la siguiente capa, la *Capa de Detección de Eventos*, un documento XML con el *Grado de Similitud* calculado y la propia salida ofrecida por el NIDS Snort en formato texto. Todo ello será lo que la siguiente capa utilice como un *Evento de Interés*. Un ejemplo de este documento podemos verlo en el listado 4.

3.1.2. Capa de Detección de Eventos

La *Capa de Detección de Eventos* es la que recibe Eventos de Interés desde la capa previa, la *Capa de Preprocesamiento*. En ella se comprobará si ese Evento de Interés, después de correlacionarlo, se supone parte de un ataque complejo definido como un grafo de ataque. A través de los eventos enviados por la *Capa de Preprocesamiento* se consigue proveer al sistema de una visión de alto nivel sobre los datos en crudo transmitidos a través del sistema monitorizado.

El formato de suceso enviado a la *Capa de Detección de Eventos* por la capa previa depende del NIDS instalado en la *Capa de Preprocesamiento*, aunque el texto en plano que obtengamos del NIDS debe ir envuelto en un documento XML junto con el *Grado de Similitud* calculado por la capa previa. Por ello, puesto que en este diseño se puede utilizar cualquier IDS de red que se quiera e incluso varios de ellos, en la primera parte de esta capa se ha situado un módulo traductor, el cual se encargará de obtener dicho documento XML y transformar de la forma adecuada el texto en plano que obtengamos del IDS de red utilizado. Este módulo, llamado *Translator to IDML*, incluye diferentes plugins para traducir cualquier posible formato dependiente de un NIDS concreto a un formato común de patrón de intrusión que sea entendible por nuestro sistema. Para esta tarea se ha elegido IDML (ver sección 2.5) como formato estándar, puesto que nos permite definir eventos, estados y transiciones entre ellos. IDML fue utilizado como formato para realizar un modelado unificado de la información en Sistemas de Detección de Intrusiones en el trabajo citado en [21]. Nuestro traductor recogerá la alerta o suceso lanzado por el *Motor de Detección* de la *Capa de Preprocesamiento*, y traducirá a IDML el texto en plano del log que nos proporciona Snort, puesto que la parte del *Grado de Interés* que acompaña al evento ya llegará con un formato apropiado.

Así pues, la entrada al *Translator to IDML* será un Evento de Interés enviado por la *Capa de Preprocesamiento* a la *Capa de Detección de Eventos*. Dicho Evento de Interés es el documento XML que incluye la información detectada por Snort en formato plano y el *Grado de Similitud* que se le añade a la salida de este NIDS. La salida del *Translator to IDML* será el *Evento de Interés normalizado* formateado de tal forma que todos los elementos de la *Capa de Detección de Eventos* sean capaces de entenderlo. Es decir, la salida incluye el *Evento de Interés* formateado en IDML en vez de estar escrito en el formato propio del NIDS subyacente que se esté utilizando. Dicho *Evento de Interés normalizado* está formado por el *Grado de Similitud* calculado y por el suceso detectado por el IDS de red que se haya utilizado en formato IDML. La Figura 10 muestra el esquema XML correspondiente a un Evento de Interés normalizado, siendo el listado 5 el DTD desarrollado correspondiente al evento definido en IDML.

Listado 5: Esquema XML que sigue la salida del módulo traductor

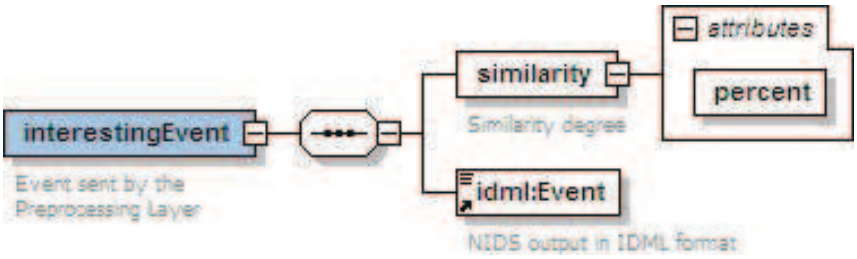

Figura 10: Definición de un Evento de Interés normalizado.

```
<xml version=''1.0'' enconding=''Big5'' ?>
<!ELEMENT interestingEvent (idml:Event, similarity)>
        <!ELEMENT idml:Event (Name, Time, Duration, Property+, Description?>
                <!ELEMENT Name (#PCDATA)>
                <!ELEMENT Time (#PCDATA)>
                <!ELEMENT Duration (#PCDATA)>
                <!ELEMENT Property (Name, Value, Description?)>
                <!ELEMENT Description (#PCDATA)>
        <!ELEMENT similarity (percent)>
                <!ELEMENT percent (#PCDATA)>
```

En la Figura 10 podemos ver varios elementos, desde el dato que se recibe en el traductor *Translator to IDML* hasta la salida compuesta por un elemento denominado *similarity*, que es el *Grado de Similitud* calculado por el *Motor de Detección*, y otro elemento denominado *idml:Event*, el cual incluye el ataque detectado por el NIDS en formato IDML.

A continuación se muestra un ejemplo de un *Evento de Interés normalizado*. Este evento se obtiene a partir del log de Snort mostrado en la tabla 2. En el log de dicha tabla se puede observar, en primer lugar, la detección de un escaneo de puertos. Para avisarnos de este suceso, Snort genera un texto en el que en primer lugar nos muestra un mensaje explicativo de lo que ha encontrado. Dicho mensaje se encuentra escrito en la regla que ha detectado el suceso, y que viene también numerada en la salida del log justo antes del mensaje. Ambas informaciones se encuentran escritas entre caracteres []. El resto de información que añade el log es la prioridad que tiene la regla que ha detectado el escaneo de puertos, una marca de tiempo, la IP de origen y destino del paquete que se ha revisado y finalmente una serie de información relativa a los campos del paquete que se ha revisado. El siguiente evento detectado por Snort ha sido una conexión TELNET, tras lo cual detecta que se ha ejecutado una orden */bin/bash/*, y finalmente se ha ejecutado el programa *netcat*. Este, además, es el orden en el que se ha ejecutado el ataque por el cual un atacante que se conecta en primer lugar mediante TELNET a la víctima, es capaz de ejecutar el programa *netcat* y dejarlo a

[**] [122:1:0] (portscan) TCP Portscan [**]
[Priority: 3]
02/28-13:34:25.802171 155.54.205.251 –> 155.54.205.80
PROTO:255 TTL:0 TOS:0x0 ID:0 IpLen:20 DgmLen:164 DF
[**] [1:9000465:0] TELNET conexion stablished [**]
[Priority: 0]
02/28-13:34:26.905926 155.54.205.251:59494 –> 155.54.205.80:23
TCP TTL:57 TOS:0x0 ID:40922 IpLen:20 DgmLen:44
******S* Seq: 0x3483BD62 Ack: 0x0 Win: 0x800 TcpLen: 24
TCP Options (1) ==> MSS: 1460
[**] [1:9000001:1] Remote shell execution (/bin/sh) [**]
[Classification: Shell execution from a remote host] [Priority: 3]
02/28-13:34:32.762408 155.54.205.251:40699 –> 155.54.205.80:23
TCP TTL:240 TOS:0x10 ID:0 IpLen:20 DgmLen:213
AP Seq: 0x8CFE1E61 Ack: 0xBF7138EC Win: 0x1700 TcpLen: 20
[**] [1:9000003:1] Command execution attempted [**]
[Classification: A suspicious command was detected] [Priority: 2]
02/28-13:34:32.762408 155.54.205.251:40699 –> 155.54.205.80:23
TCP TTL:240 TOS:0x10 ID:0 IpLen:20 DgmLen:213
AP Seq: 0x8CFE1E61 Ack: 0xBF7138EC Win: 0x1700 TcpLen: 20

Tabla 2: Log obtenido desde Snort.

la escucha de conexiones desde otros clientes por el puerto deseado por el atacante, con lo que este puede conectarse a la víctima sin ningún problema.

La traducción de este evento a nuestro formato intermedio es la mostrada en el listado 6:

Listado 6: Traducción de un evento a formato IDML

```
<xml version=''1.0'' enconding=''Big5'' ?>
<interestingEvent>
        <idml:Event>
                <Name>TCP Portscan</Name>
                <Time>02/28-13:34:25</Time>
                <Duration></Duration>
                <Property>
                        <Name>IP Origen</Name>
                        <Value>155.54.205.251</Value>
```

```
                </Property>
                <Property>
                        <Name>IP Destino</Name>
                        <Value>155.54.205.80</Value>
                </Property>
                <Description>TCP port scan</Description>
        </idml:Event>
        <similarity>
                <percent>100</percent>
        </similarity>
</interestingEvent>

<interestingEvent>
        <idml:Event>
                <Name>TELNET conexion</Name>
                <Time>02/28-13:34:26</Time>
                <Duration></Duration>
                <Property>
                        <Name>IP Origen</Name>
                        <Value>155.54.205.251</Value>
                </Property>
                <Property>
                        <Name>Puerto Origen</Name>
                        <Value>59494</Value>
                </Property>
                <Property>
                        <Name>IP Destino</Name>
                        <Value>155.54.205.80</Value>
                </Property>
                <Property>
                        <Name>Puerto Destino</Name>
                        <Value>23</Value>
                </Property>
                <Description>TELNET conexion stablished</Description>
        </idml:Event>
        <similarity>
                <percent>100</percent>
        </similarity>
</interestingEvent>

<interestingEvent>
```

```
        <idml:Event>
                <Name>RemoteShellExecution</Name>
                <Time>02/28-13:34:32</Time>
                <Duration></Duration>
                <Property>
                        <Name>IP Origen</Name>
                        <Value>155.54.205.251</Value>
                </Property>
                <Property>
                        <Name>Puerto Origen</Name>
                        <Value>40699</Value>
                </Property>
                <Property>
                        <Name>IP Destino</Name>
                        <Value>155.54.205.80</Value>
                </Property>
                <Property>
                        <Name>Puerto Destino</Name>
                        <Value>23</Value>
                </Property>
                <Description>Remote shell execution (/bin/sh)</Description>
        </idml:Event>
        <similarity>
                <percent>100</percent>
        </similarity>
</interestingEvent>

<interestingEvent>
        <idml:Event>
                <Name> CommandExecution </Name>
                <Time>2/28-13:34:32</Time>
                <Duration></Duration>
                <Property>
                        <Name>IP Origen</Name>
                        <Value>155.54.205.251</Value>
                </Property>
                <Property>
                        <Name>Puerto Origen</Name>
                        <Value>40699</Value>
                </Property>
                <Property>
```

```
                <Name>IP Destino</Name>
                <Value>155.54.205.80</Value>
            </Property>
            <Property>
                <Name>Puerto Destino</Name>
                <Value>23</Value>
            </Property>
            <Description> Command execution attempted</Description>
        </idml:Event>
        <similarity>
            <percent>100</percent>
        </similarity>
</interestingEvent>
```

En cuanto a los componentes que aparecen en esta capa son dos. La *Base de datos de Eventos* y el *Motor de Eventos*. El primero es una base de datos interna que provee de patrones basados en eventos (i.e. grafos de ataque) al *Motor de Eventos*. Esta es la parte del sistema al cual llegan, finalmente, los eventos sospechosos detectados por el NIDS de la *Capa de Preprocesamiento*. En dicha parte de la arquitectura se estudian dichos eventos para comprobar si realmente suponen un riesgo para el sistema o no.

Todos los flujos de datos internos de esta capa están representados usando para ello el mismo estándar IDML. Este formato permite, además de lo expuesto anteriormente, expresar tanto las transiciones entre los estados de los grafos de ataque, lo cual permite definir cada fase del proceso de intrusión, como los propios estados de dichos grafos de ataque. En esta capa, al flujo de datos que sale de ella lo hemos llamado *Incidente* (*Incidents* en la figura 9). Dicho *Incidente* estará formado por un documento XML en el que se incluirá:

- Un identificador del incidente. Este identificador corresponde con el identificador del grafo de ataque detectado.
- De dónde procede dicho incidente (dirección IP origen y puerto origen).
- Cuál es el destino del mismo (dirección IP destino y puerto destino).
- La hora exacta en la que se ha producido el incidente.
- El *Grado de Credibilidad* del mismo (explicado en una subsección posterior).
- Un mensaje opcional con el que se explica el incidente.
- Finalmente un elemento XML que puede ser utilizado para extender dicho esquema XML y poder incluir lógica propia en un futuro.

Este formato lo podemos observar en la figura 11.

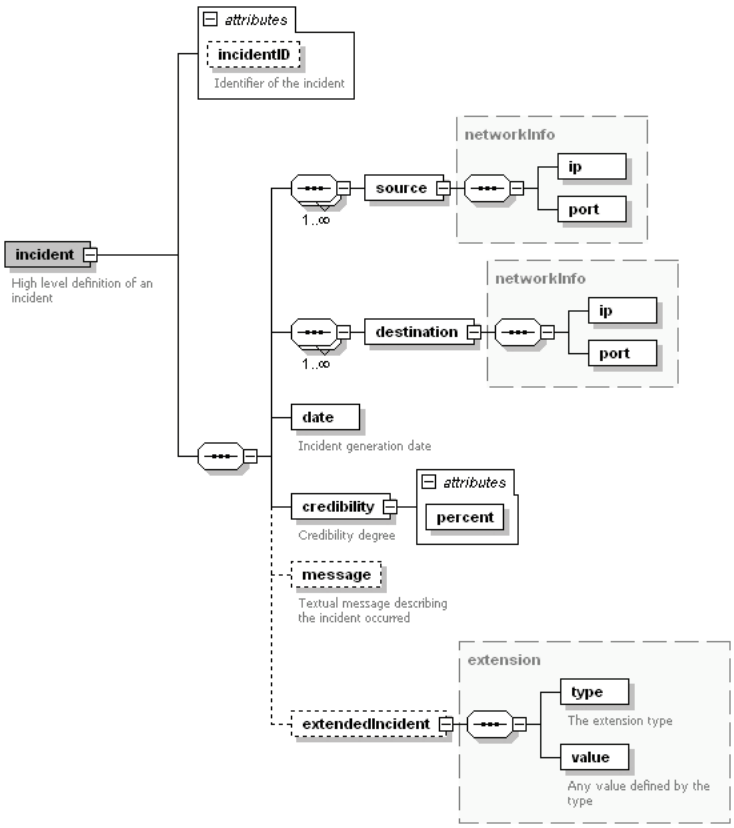

Figura 11: Definición de alto nivel de un incidente.

Base de datos de Eventos

Se corresponde a la *DB with Events* que podemos observar en la figura 9. Es la base de datos en la que se almacenan los eventos que dan lugar a un ataque o problema de seguridad en la red monitorizada. Los eventos en esta base de datos se almacenan en forma de patrones de eventos, utilizando para ello los *Grafos de Ataque*. Como hemos dicho anteriormente, estos grafos están definidos en formato IDML para facilitar el mapeo entre los *Eventos de Interés* recibidos (elemento *idml:Event* en Figura 10) y los eventos almacenados en ella. Por lo tanto, a partir de estos eventos, el siguiente elemento de esta capa de la arquitectura, el *Motor de Eventos*, es capaz de decidir si se ha llevado a cabo un ataque complejo en el sistema o no.

Cada grafo de ataque almacenado en la *Base de datos de eventos* es etiquetado con un identificador, el cual se usará en la siguiente fase para identificar qué patrones de ataque ha seguido el atacante en el sistema monitorizado. Usando el mismo identificador, podemos agrupar más de un grafo de ataque para almacenar grafos del mismo tipo. Por ejemplo, todos los grafos de ataque que representen diferentes ataques de un cierto tipo de virus se pueden agrupar bajo el mismo identificador. Esto se puede usar después para asignar la misma decisión a todos los grafos de ataque que estén representados en el mismo grupo, evitando así almacenar información redundante en el sistema.

Motor de Eventos

Se corresponde al *Event Engine* que podemos observar en la figura 9. Es la parte del sistema a la que llegan los eventos desde la *Capa de Preprocesamiento*. A partir de los eventos *idml:Event* y de los *Grafos de Ataque* almacenados en la *Base de datos de Eventos*, decide si se está produciendo un ataque complejo o no. Incluye información de estado más significativa que la que encontrábamos en la *Capa de Preprocesamiento*, ya que además de incluir el estado actual del sistema, tiene almacenada información sobre eventos pasados, con lo que dispone de una información más global en el tiempo que la capa inferior.

El *Motor de Eventos* mantiene en la memoria la siguiente información para cada usuario sospechoso:

1. La dirección IP del atacante.
2. La dirección IP de la máquina que está siendo atacada.
3. Una lista con los estados actuales en los cuales se encuentra el atacante (un mismo atacante puede estar llevando a cabo más de un ataque al mismo tiempo). Cada estado actual almacena la siguiente información:
   - El identificador del grafo de ataque el cual está siguiendo o llevando a cabo el atacante.

- Una lista con los próximos estados a los cuales debe moverse el atacante. En este sentido, la arquitectura puede anticiparse descubriendo si el siguiente estado es un estado final y así poder reaccionar tan rápido como sea posible.
- Un valor de probabilidad acumulado, llamado *Grado de Credibilidad*, para indicar el porcentaje de coincidencia que tienen los diferentes pasos que está ejecutando el atacante con respecto a los pasos definidos en los diferentes grafos de ataque por los expertos en seguridad del sistema. Este valor es explicado en la sección 3.1.2.

Cuando se recibe un *Evento de Interés* desde la *Capa de Preprocesamiento* (ver Figura 10), el *Motor de Eventos* busca en su memoria si el atacante tiene algún estado actual dentro del sistema. Es decir, el *Motor de Eventos* comprueba si el posible atacante acaba de empezar a interactuar con el sistema monitorizado o si por el contrario ya está tiempo enviando paquetes y realizando acciones comprometidas para el sistema. Para esto, este módulo hace un mapeo entre la dirección IP que contiene el evento recibido y la lista de direcciones IP almacenadas en memoria. Si existe una coincidencia, el *Motor de Eventos* chequea si el atacante se ha movido al siguiente estado predefinido dentro del grafo de ataque en el que está encuadrado, haciendo para ello un mapeo entre el documento IDML del suceso recibido desde la *Capa de Preprocesamiento*, y traducido por el *Translator to IDML*, y la lista de estados siguientes a los cuales es posible que acceda el atacante. Si no existe ningún estado al que el atacante pueda ir con el nuevo evento recibido, el evento es descartado, puesto que no supone ningún riesgo para el sistema. De otro modo, el atacante debería haber actualizado su estado actual al siguiente estado esperado, dentro del grafo de ataque. Así, el *Motor de Eventos* deberá realizar con cada *Evento de Interés normalizado* recibido las siguientes acciones. Por un lado, deberá descargar la lista con los siguientes estados desde la *Base de datos de Eventos* y, por otro lado, deberá calcular el *Grado de Credibilidad* acumulado.

Cuando esta capa detecta un *Evento de Interés*, es decir, cuando el atacante ha conseguido alcanzar el estado final dentro del grafo de ataque que ha recorrido mientras interactuaba con el sistema monitorizado, esta capa envía un *Incidente* (*Incidents* en la figura 9) a la siguiente, la *Capa de Decisión*, etiquetando dicho incidente como *"ataque"* e incluyendo el identificador del patrón de ataque como identificador de dicho incidente. Así esta capa puede reaccionar y tratar el problema tan pronto como le llegue un incidente. La figura 11 muestra la definición de este incidente de alto nivel, el cual incluye la información necesaria para decidir la reacción específica para poder ejecutarla más tarde. Hay que notar que el elemento *credibility* transportará un *Grado de Credibilidad* el cual representará la certeza que la arquitectura tiene sobre la posibilidad de que el ataque detectado sea realmente un ataque o no. En el listado 7 podemos observar un ejemplo de posible incidente que esta capa enviaría a la *Capa de Decisión*. En ella vemos que el incidente se componen básicamente de una dirección IP origen con el puerto desde el que se recibe el evento, y de una IP destino y el puerto al que va dirigido. Asimismo se incluye el *Grado de Credibilidad* que ha sido asignado a dicho

evento. El incidente también incluye una marca de tiempo para disponer de información temporal sobre lo ocurrido.

Listado 7: Incidente enviado a la Capa de Decisión

```
<incident incidentID=''identificadorIncidente''>
     <source>
          <ip>155.51.205.80</ip>
          <port>17568</port>
     </source>
     <destination>
          <ip>64.123.110.37</ip>
          <port>25634</port>
     </destination>
     <date>27/08/2008 17:00</date>
     <credibility percent=''100''/>
</incident>
```

Grado de Credibilidad

Al igual que hemos propuesto en la *Capa de Preprocesamiento*, en la que hemos calculado el *Grado de Similitud*, para esta capa proponemos también asignar más flexibilidad a través de la estimación de un nuevo valor probabilístico, al cual llamamos *Grado de Credibilidad*. Este valor nos da un porcentaje de la exactitud que el atacante ha tenido sobre cada uno de los pasos que ha seguido durante su ataque al sistema. Es decir, es el nivel de acierto que ha realizado el atacante a la hora de realizar los pasos predichos por el grafo de ataque que ha seguido dentro del sistema.

El *Grado de Credibilidad* se calcula mediante el nivel de coincidencia entre el estado actual y el evento enviado por el *Motor de Detección*, ambos en formato IDML. Además también se utiliza el *Grado de Similitud* calculado en la *Capa de Preprocesamiento* y el valor acumulado del *Grado de Credibilidad* para el atacante en el grafo de ataque en el que se encuentre en la actualidad. Esto lo podemos ver en la ecuación 1.

$$GC = GC_{acumulado} + GS + NivelCoincidenciaEstadoActual \quad (1)$$

donde GC es el *Grado de Credibilidad* actual sobre el posible ataque, $GC_{acumulado}$ es el *Grado de Credibilidad* que teníamos antes del último paso realizado, GS es el *Grado de Simil-*

itud que tiene el último paso realizado por el atacante y $NivelCoincidenciaEstadoActual$ es el nivel de coincidencia entre el estado actual y el evento enviado por el *Motor de Detección*.

3.1.3. Capa de Decisión

La *Capa de Decisión* recibe la resolución tomada por la *Capa de Detección de Eventos*, en forma de *Incidente*, y ejecuta la acción apropiada para reaccionar ante el ataque al que se está sometiendo a una o varias máquinas del sistema monitorizado. Esta capa está compuesta por la *Base de datos con la Lista de Decisiones (DB with Decision List)* y el *Motor de Decisión (Decision Engine)*. La base de datos almacena la información necesaria para saber cómo debe reaccionar el sistema cuando se recibe un incidente etiquetado como "ataque". El *Motor de Decisión* determina qué conjunto de acciones se deberían llevar a cabo en el sistema, haciendo para ello un mapeo entre el incidente que ha recibido de la *Capa de Detección de Eventos* y la lista de acciones que están almacenadas en la base de datos (la *Decision List* de la figura 9).

Las decisiones son tomadas en base a los elementos *incidentID* y *credibility* que se encuentran almacenados en el incidente recibido (ver Figura 11). Entre las posibles acciones a realizar, el *Motor de Decisión* puede enviar esta información al administrador, puede bloquear la IP del atacante, y así sucesivamente puede realizar otras acciones o varias de ellas a la misma vez. El tipo de acciones que pueden llevarse a cabo en el sistema para solucionar los problemas causados por el atacante (o al menos evitar que se propaguen), pueden ser diferentes según el *Grado de Credibilidad* calculado en la capa anterior y enviado a esta capa dentro del elemento credibility de la Figura 11. Por ejemplo, si la reacción asociada al incidente recibido fuera bloquear la IP del atacante, pero la arquitectura no está completamente segura de que el ataque se ha realizado realmente (puesto que posee un grado de credibilidad del 75 %), es el *Motor de Decisión* quien debería decidir informar al administrador en vez de aplicar la correspondiente regla de filtrado directamente en el cortafuegos.

3.2. Resumen de la Arquitectura

A modo de resumen de la arquitectura presentada mostramos un diagrama de secuencia, que podemos ver en la Figura 12 que muestra las partes en las que se descompone la arquitectura presentada y la interacción entre todas las capas. Así se podrá observar de una forma sencilla y rápida el flujo que recorre esta arquitectura. Además, facilitará el entendimiento de cada una de las capas, y se verá el momento en el que cada una de ellas pasa a formar parte del camino de datos.

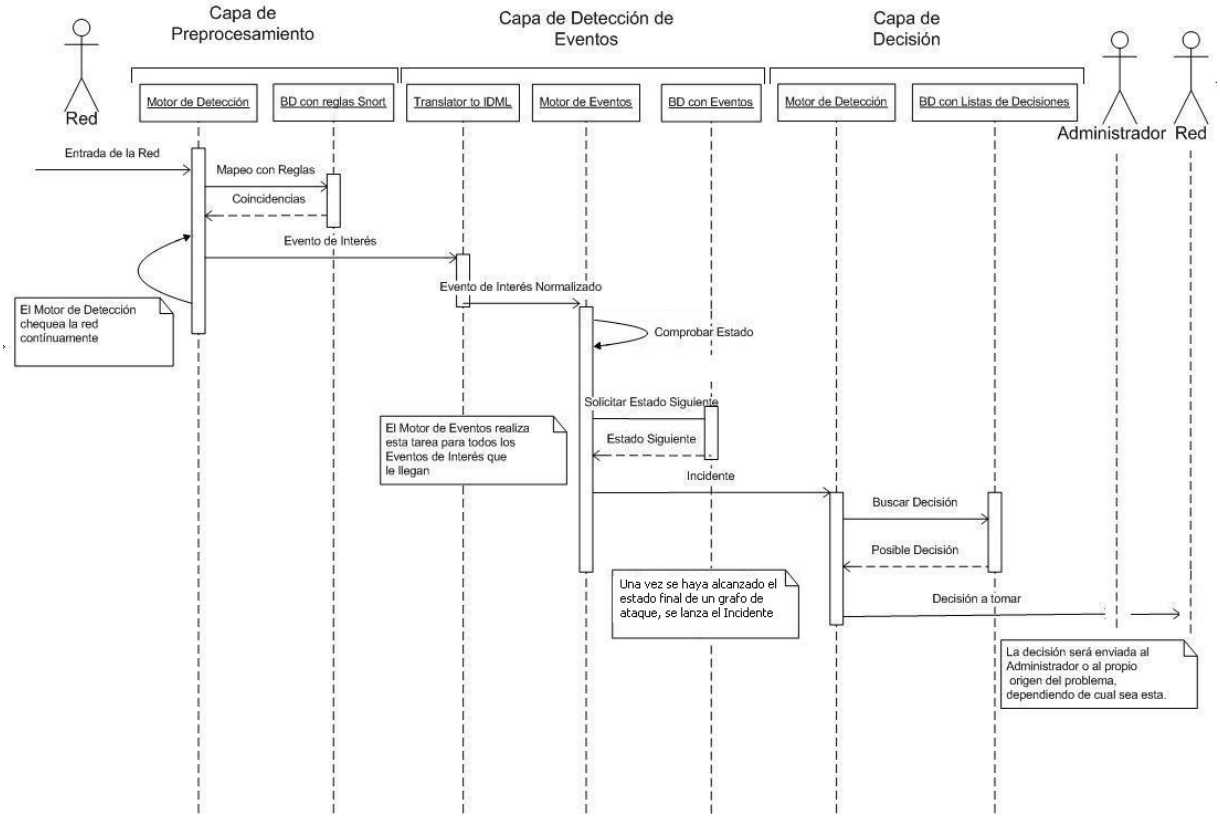

Figura 12: Diagrama de secuencia de la arquitectura

3.3. Ejemplo ilustrativo

En esta sección mostraremos un caso de uso en el cuál la arquitectura propuesta se usa para realizar una prueba de concepto sobre un ataque complejo. Este ataque está compuesto por cuatro pasos, los cuales incluyen una conexión TELNET y un conjunto de comandos que pueden causar problemas de seguridad en cualquier sistema. Aunque TELNET ya prácticamente no se usa debido a los riesgos de seguridad que ocasiona, es una herramienta útil para proporcionar un flujo de datos en claro para que éste sea inspeccionado por un motor de detección basado en reglas.

Llamaremos “Atacante” a la máquina de la persona que está intentando atacar al sistema, y “Víctima” a la máquina objetivo del ataque. También supondremos que este último tiene instalado y configurado Snort en la *Capa de Preprocesamiento* para poder proveer a nuestro *Motor de Eventos* todos los eventos que sean interesantes para su estudio.

La configuración básica de las máquinas utilizadas, mostrada en la Figura 13, es la sigu-

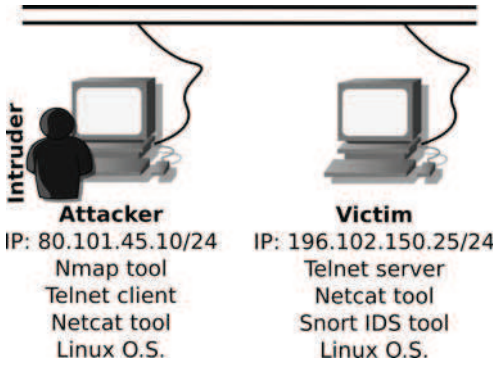

Figura 13: Configuración del caso de uso.

iente:

- Atacante: S.O. Linux, herramienta nmap para ejecutar un escaneo de puertos, un cliente TELNET y la herramienta netcat. Esta herramienta permite abrir puertos TCP/UDP en una máquina (quedando netcat a la escucha), asociar una shell a un puerto en concreto (para conectarse por ejemplo a MS-DOS o al intérprete bash de Linux remotamente) y forzar conexiones UDP/TCP (útil por ejemplo para realizar rastreos de puertos o realizar transferencias de archivos bit a bit entre dos equipos).
- Víctima: S.O. Linux, un demonio de TELNET corriendo en el puerto 23 para permitir conexiones remotas, la herramienta netcat y Snort para procesar todos los paquetes enviados a la Víctima.

El comportamiento del atacante está basado en los siguientes pasos:

1. El Atacante hace un escaneo de puertos contra la Víctima con el ánimo de descubrir los servicios que éste tenga disponibles. El Atacante encontrará escuchando el puerto 23, correspondiente al servicio de TELNET.
2. El Atacante se conectará a la víctima vía TELNET utilizando para ello una cuenta de root. Supondremos que el Atacante ha estado realizando sniffing de la red y ha encontrado una conexión previa en la que éste ha obtenido el usuario y el password.
3. El Atacante ejecuta un comando para dejar una puerta trasera en la Víctima usando la herramienta netcat. El comando que se ejecuta deja a la herramienta netcat escuchando en el puerto que el Atacante desee y le ordena ejecutar un shell cuando encuentre una conexión desde un cliente.

4. El Atacante se conecta a la Víctima usando la puerta trasera creada en la máquina afectada.

En la *Capa de Preprocesamiento*, dentro de la *Base de datos con las reglas de Snort*, hemos añadido algunas reglas nuevas a Snort para poder observar mejor los diferentes comandos que el Atacante puede ejecutar. Estas reglas, mostradas en la tabla 3, intentan detectar los siguientes eventos de interés:

- Cualquier paquete de conexión (con el flag Syn activado) dirigido al puerto 23 de la Víctima, lo que se controla con la primera regla de la tabla 3.
- Cualquier paquete con el contenido "/bin/bash" o "/bin/sh" para detectar la ejecución remota de un shell, que corresponde a la segunda y tercera regla de la tabla 3.
- Cualquier paquete que contenga "nc" para inspeccionar las ejecuciones de netcat, siendo la cuarta regla de la tabla 3 la que controla este suceso.

alert tcp any any –> $TELNET_SERVERS 23 (msg: TELNET connection established; flags:S,12;sid:9000465;*similarity:100*)
alert tcp any any –> $TELNET_SERVERS 23 (msg: Remote shell execution (/bin/sh); flow:to_server, established; content:/bin/sh; classtype:remote-shell-execution; sid:9000001; rev:1;*similarity:100*)
alert tcp any any –> $TELNET_SERVERS 23 (msg: Remote shell execution (/bin/bash) flow:to_server, established; content:/bin/bash; classtype:remote-shell-execution; sid:9000002; rev:1;*similarity:100*)
alert tcp any any –> $TELNET_SERVERS 23 (msg: Command execution attempted; flow:to_server, established; content:nc; classtype:dangerous-command; sid:9000003; rev:1;*similarity:100*)

Tabla 3: Nuevas Reglas definidas para Snort.

Las reglas para detectar el escaneo de puertos no han sido incluidas puesto que Snort define las suyas propias por defecto.

Cuando se ejecuta alguna de las reglas descritas, Snort la envía al *Motor de Eventos* el cual la analizará para chequear si ese evento es parte de un ataque complejo o no.

Para tener una idea clara sobre el flujo de datos y la funcionalidad de cada elemento de la arquitectura, mostramos la secuencia de comandos ejecutados por el Atacante, tanto en

su propia máquina como en la máquina de la Víctima (a través de la conexión TELNET). Mostraremos también el flujo de datos generado por el propio sistema.

La evolución del *Motor de Eventos*, perteneciente a la *Capa de Detección*, es la mostrada en la Figura 14.

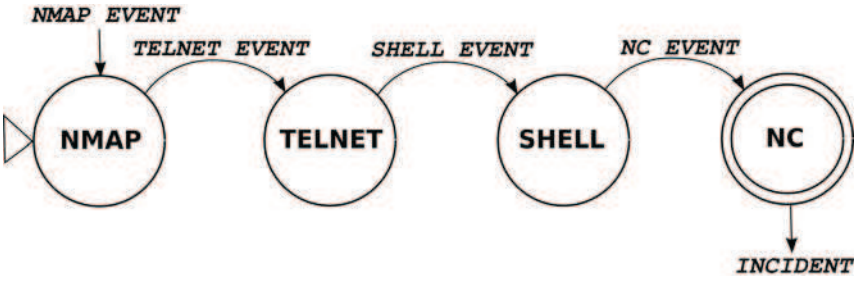

Figura 14: Transacciones del Motor de Eventos.

1. *[Atacante]# nmap -sS -sV 196.102.150.25*: El propósito de este comando es obtener un listado con los servicios disponibles en la máquina objetivo. Así el Atacante puede descubrir que el servicio TELNET está disponible en el puerto 23. En nuestra arquitectura, Snort hará un mapeo entre el paquete de red que ha llegado generado por este comando contra las reglas definidas en su base de datos y, en este ejemplo, encontrará una ocurrencia. Entonces Snort enviará este suceso de interés al *Motor de Eventos*. Ya que este evento es definido como estado inicial en la lista de patrones almacenada en la *Base de datos con los Eventos*, el primer estado en Figura 14, el *Motor de Eventos* lo mantendrá en memoria como el estado actual del Atacante. La arquitectura esperará a que lleguen nuevos eventos para situarlos como estados posteriores a la ejecución de nmap.

2. *[Atacante]# telnet 196.102.150.25*: El Atacante se conecta al servidor TELNET usando la información previa que ha escuchado sobre el usuario y el password. Snort analiza esos paquetes de red y reporta un evento nuevo al *Motor de Eventos*, el cual chequeará si el Atacante se ha situado en el siguiente estado dentro del grafo de ataque. El *Motor de Eventos* actualiza su información interna para indicar que el Atacante ha alcanzado el segundo estado en la Figura 14. El Atacante ha accedido a la Víctima.

3. *[Víctima]# nc -l -e /bin/bash -p 25634*: El Atacante deja la herramienta netcat escuchando en el puerto 25634 y le ordena ejecutar un shell cuando encuentre una conexión desde un cliente. Como en los pasos previos, este evento será reportado al

Motor de Eventos, el cual actualizará el estado actual de nuevo para indicar que el Atacante se encuentra en el tercer estado del grafo de ataque. El Atacante ha conseguido dejar escuchando un troyano dentro de la Víctima.

4. *[Atacante]# nc -vv 196.102.150.25 25634*: Por último, el Atacante accede al shell remoto a través de la herramienta netcat. Cuando el *Motor de Eventos* actualice el estado actual al siguiente, éste sabrá que este estado es un estado final y, en consecuencia, el *Motor de Decisión* deberá ser informado a este respecto. El *Motor de Decisión* chequeará la acción correcta a realizar buscando una coincidencia en la *Base de datos con la Lista de Decisiones*, y ejecutará la acción asociada a ésta (por ejemplo, bloquear la dirección IP del Atacante o alertar al Administrador).

3.4. Implementación del Prototipo

En esta subsección se presenta una pequeña implementación del prototipo que sirva como muestra y ejemplo de lo presentado en la arquitectura de este trabajo.

La implementación de este prototipo se ha realizado en el lenguaje de programación *Java*, en su versión 1.6.0_04, utilizando para ello el programa de desarrollo *Netbeans*, siendo la versión de este IDE la 6.0.1. El Sistema Operativo elegido para realizar las pruebas del prototipo ha sido *Ubuntu 7.10 Gutsy Gibbon*, puesto que es un entorno en el que es muy sencillo utilizar el NIDS *Snort*, siendo la versión utilizada la 2.7.0. Este NIDS se ha ejecutado en la misma máquina que actúa como Víctima, puesto que la única intención de este prototipo es mostrar la utilidad de la arquitectura que hemos desarrollado.

En la figura 15 se muestra un diagrama de bloques con cada una de las clases que conformarán el prototipo de la arquitectura presentada.

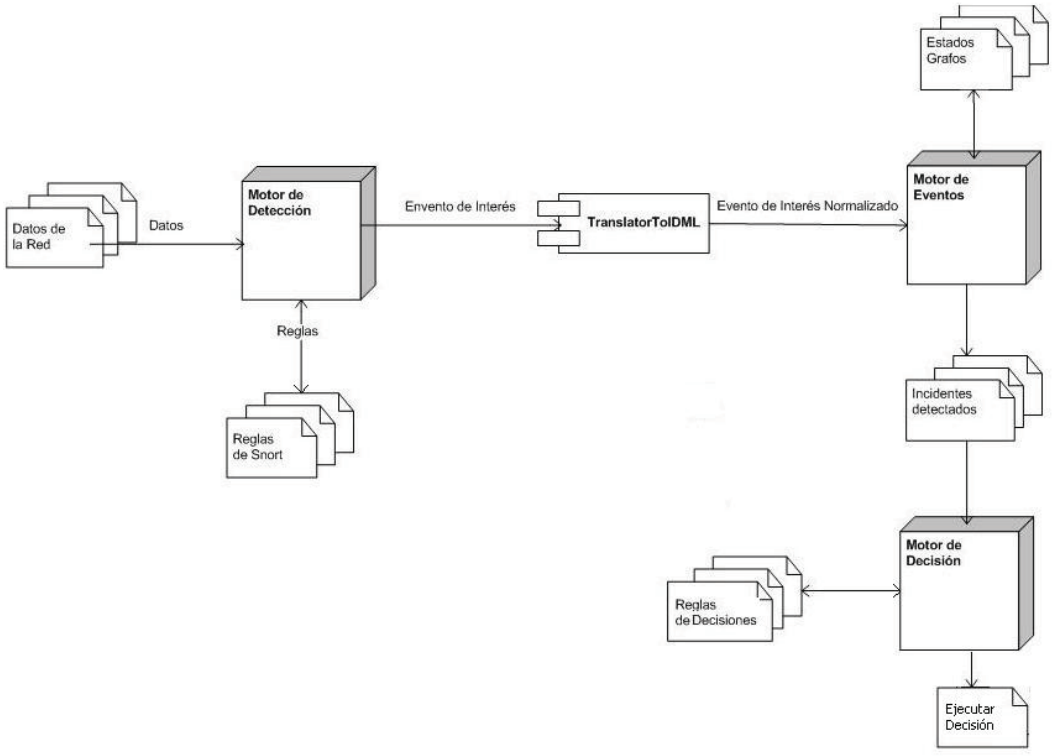

Figura 15: Diagrama de Bloques del prototipo

En la figura 16 mostramos la pantalla principal del prototipo en la cuál se puede observar

cómo se va chequeando todo lo que Snort nos envía a la *Capa de Detección de Eventos*. Además se puede ver el estado actual en el que se encuentra el ataque que se está registrando en ese mismo momento.

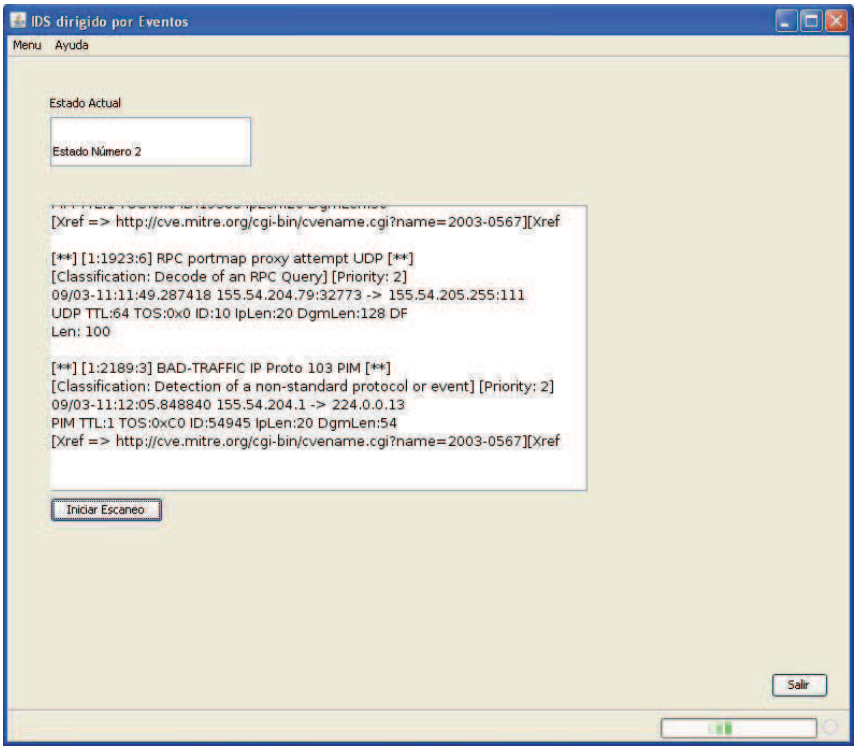

Figura 16: Pantalla principal del prototipo

En la figura 17 mostramos la ventana emergente que aparece cada vez que un Incidente es encontrado en el sistema y reportado a la *Capa de Decisión*.

A la hora de incluir el *Grado de Similitud* en las reglas de Snort, el motor de detección de este NIDS no acepta de este modo un nuevo campo, puesto que Snort no está programado para recoger ese nuevo campo. En la aproximación que mostramos como prototipo, este campo no está codificado. Por ello, en tiempo de ejecución no se hace la comparación de la que hablábamos en el apartado 3.1.1, entre el *Grado de Similitud* que la regla tiene por defecto y el que se calcula. Es por esto que para la implementación de este prototipo hemos optado por asumir que este campo es siempre el 100 %.

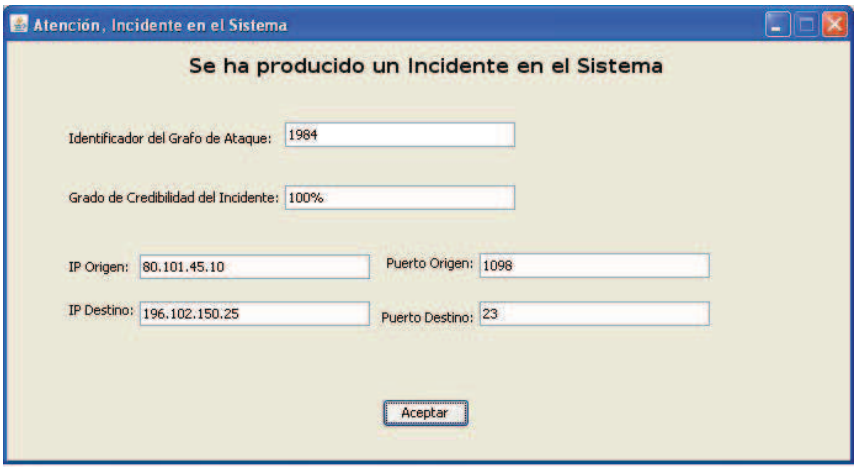

Figura 17: Ventana emergente de aviso de incidente

4. Conclusiones y vías futuras

En este trabajo se ha presentado una arquitectura dirigida por eventos para un Sistema de Detección de Intrusiones basado en patrones. Se han estudiado varias de las bases de datos de vulnerabilidades Open Source más importantes del mercado, así como el sistema de detección de intrusiones basado en red que constituye el estándar de facto dentro de este campo, Snort. También se han estudiado las técnicas de correlación en general, y los grafos de ataque en particular. En este trabajo se ha hecho uso, además, de IDML para la definición de patrones de intrusión.

La arquitectura presentada es una arquitectura dirigida por eventos de alto nivel que calcula dos valores de probabilidad sobre los ataques, llamados *Grado de Similitud* y *Grado de Credibilidad*. Estos grados representan, por un lado, la similitud entre un suceso actual y uno almacenado en la base de datos de los NIDS utilizados (*Grado de Similitud*, el cual está basado en la detección de eventos), lo cual nos permite manejar posibles mutaciones o variaciones de ataques ya conocidos. Por otro lado, la credibilidad o confianza en que realmente esté sucediendo un ataque sobre el sistema monitorizado (*Grado de Credibilidad*, que está basado en la detección de patrones). Ambos valores aportan un nivel de flexibilidad mayor que el que poseen los IDS actuales, permitiendo reconocer posibles variaciones de los ataques ya conocidos sin necesidad de tener que incluir esas variaciones implícitamente en las bases de datos de los diferentes motores.

Entre las vías futuras que sugerimos para la ampliación de este trabajo encontramos la utilización de diferentes NIDS dentro de la *Capa de Preprocesamiento*, para poder obtener mayor información y más veraz sobre lo que está ocurriendo en cierto sistema. Otra de las

ampliaciones podría ser la mejora y afinación en el cálculo tanto del *Grado de Similitud* como del *Grado de Credibilidad* en las dos primeras capas de la arquitectura.

Otra posible vía futura es el cálculo del número de falsos positivos aportados por esta arquitectura.

También es interesante incluir la necesidad de adaptar el NIDS elegido, en este caso Snort, para que sea capaz de admitir el nuevo campo de similitud, *similarity*, incluido en sus reglas.

Otra de las vías futuras es la de la generación automática de los grafos de ataque. Ésta es un punto muy importante que actualmente se está estudiando. Por ejemplo, las bases de datos de vulnerabilidades podrían proporcionar directamente esos grafos de ataque.

Finalmente, otra de las posibles vías de ampliación sería la de introducir en la arquitectura la información aportada por las bases de datos de vulnerabilidades, puesto que éstas cada vez se están haciendo más importantes e incluyen información cada vez más útil para localizar posibles vulnerabilidades dentro de cualquier sistema.

Referencias

[1] Stefan Axelsson. Intrusion detection systems: A survey and taxonomy. Technical Report 99–15, Chalmers University, 2000.

[2] The MITRE Corporation. Introduction to the oval language, version 5.0. *http://oval.mitre.org*, 2008.

[3] D. Curry, H. Debar, et al. Intrusion detection message exchange format data model and extensible markup language (xml) document type definition. *IDWG, February*, 2002.

[4] Chris Abernethy Daniel B. Cid. Ossec, open source host-based intrusion detection system. Web, September 2008.

[5] Hervé Debar, Marc Dacier, and Andreas Wespi. Towards a taxonomy of intrusion-detection systems. *Computer Networks*, 31(9):805–822, 1999.

[6] DE Denning. An intrusion-detection model. *Software Engineering, IEEE Transactions on*, pages 222–232, 1987.

[7] C.F. Endorf and J. Mellander. *Intrusion Detection & Prevention*. McGraw-Hill Osborne Media, 2004.

[8] AO Freier, P. Karlton, and PC Kocher. Secure socket layer 3.0. *IETF draft, November*, 1996.

[9] Diego González Gómez. *Sistemas de Detección de Intrusiones*. http://www.dgonzalez.net, 2003.

[10] Suvajit Gupta and Joel Winstead. Using attack graphs to design systems. *IEEE Security and Privacy*, 5(4):80–83, 2007.

[11] Joshua Haines, Dorene Kewley Ryder, Laura Tinnel, and Stephen Taylor. Validation of sensor alert correlators. *IEEE Security and Privacy*, 01(1):46–56, 2003.

[12] Yao-Tsung Lin, Shian-Shyong Tseng, and Shun-Chieh Lin. An intrusion detection model based upon intrusion detection markup language. *Journal of Information Science and Engineering*, 17(6):899–919, 2001.

[13] Fort George G. Meade. A guide to understanding audit in trusted systems, 1987.

[14] P.Ñing, Y. Cui, and D.S. Reeves. Constructing attack scenarios through correlation of intrusion alerts. *Proceedings of the 9th ACM conference on Computer and communications security*, pages 245–254, 2002.

[15] P.Ñing, Y. Cui, D.S. Reeves, and D. Xu. Techniques and tools for analyzing intrusion alerts. *ACM Transactions on Information and System Security (TISSEC)*, 7(2):274–318, 2004.

[16] P.Ñing and D. Xu. Learning attack strategies from intrusion alerts. *Proceedings of the 10th ACM conference on Computer and communications security*, pages 200–209, 2003.

[17] NVD. National vulnerability data base. Web, May 2008.

[18] OSVDB. Open source vulnerabilities data base. Web, May 2008.

[19] Cynthia Phillips and Laura Painton Swiler. A graph-based system for network-vulnerability analysis. In *NSPW '98: Proceedings of the 1998 workshop on New security paradigms*, pages 71–79, 1998.

[20] The Snort Project. *Snort Users Manual v2.6.1*, April 2007.

[21] Miguel A. Hernández Ruiz. Modelo unificado de información para sistemas de detección de intrusos. Master's thesis, Facultad de Informática, Universidad de Murcia, 2006.

[22] M. Sebring, E. Shellhouse, M. Hanna, and R. Whitehurst. Expert systems in intrusion detection: A case study. *Proceedings of the 11th National Computer Security Conference, October*, 1988.

[23] SE Smaha, T.A.S. Inc, and TX Austin. Haystack: an intrusion detection system. *Aerospace Computer Security Applications Conference, 1988., Fourth*, pages 37–44, 1988.

[24] PreludeIDS Technologies. Prelude open source intrusion detection system. Web, September 2008.

[25] S.J. Templeton and K. Levitt. A requires/provides model for computer attacks. *Proceedings of the 2000 workshop on New security paradigms*, pages 31–38, 2001.

[26] W. Tener. Discovery: An expert system in the commercial data security environment. *Security and Protection in Information Systems: Proceedings of the Fourth Ifip Tc 11 International Conference on Computer Security, Ifip/Sec'86, Monte Carlo, Monaco, 2-4 December, 1986*, 1989.

[27] L. Wang, A. Singhal, and S. Jajodia. Toward measuring network security using attack graphs. *Proceedings of the 2007 ACM workshop on Quality of protection*, pages 49–54, 2007.

A. Reglas en Snort. Significado, formato y campos

A.1. Introducción

En este anexo se mostrarán todos los atributos con los que se pueden formar las reglas de detección de Snort, a excepción de aquellos que la propia documentación de Snort afirma no utilizar en la actualidad.

Se mostrará después de cada atributo, su descripción y su formato.

Snort utiliza el siguiente formato de reglas:

Cabecera (Opciones)

O más específicamente:

Acción Protocolo IPOrigen PuertoOrigen -> IPDestino PuertoDestino (Opciones)

Los campos de la cabecera aparecen en todas las reglas de Snort. El resto son opcionales, y algunos de ellos pueden aparecer varias veces seguidas.

A.2. Cabecera de las reglas

La cabecera de la regla contiene la información que define el ¿Quién?, ¿Dónde? y ¿Qué? de un paquete, así como también que hacer en caso de que sea mostrado un paquete con todas las cualidades indicadas en la regla.

Acciones de las reglas

El primer campo en una regla es la acción de la misma. Ésta le indica a Snort que hacer cuando encuentra un paquete que concuerda con los criterios de la regla. Existen cinco acciones disponibles por defecto en Snort: alert, log, pass, activate y dynamic.

- alert: genera una alerta usando el método de alerta seleccionado y luego registra el paquete.

- log: registra el paquete.
- pass: ignora el paquete.
- activate: alerta y luego enciende otra regla dinámica.
- dynamic: permanece en estado de ocio hasta que sea activada por una regla de activate y luego funciona como una regla de registro.

También se pueden definir tipos de reglas propias y asociar uno o más plugins de salida con ellas. Luego se puede usar los tipos de reglas como acciones en las reglas de Snort.

Protocolos

El próximo campo en la regla es el protocolo. Existen cuatro protocolos que actualmente analiza para detectar comportamientos sospechosos: TCP, UDP, ICMP e IP. En el futuro se incluirán más como lo son: ARP, IGRP, GRE, OSPF, RIP, IPX, entre otros.

Direcciones IP

La siguiente porción de la cabecera de la regla trata de la información de la dirección IP y puerto para una regla dada. La palabra clave any puede ser usada para definir cualquier dirección. Snort no tiene un mecanismo para suministrar operaciones de búsqueda del nombre del host para los campos de dirección IP en el archivo de reglas. Las direcciones están formadas por una dirección IP un una línea numérica y un bloque CIDR. El bloque CIDR indica la máscara de red que sería aplicada para la dirección de la regla y a cualquier paquete entrante que es probado contra la misma. Por ejemplo, la combinación dirección/CIDR 192.168.1.0/24 significaría el bloque de direcciones desde la 192.168.1.1 hasta la 192.168.1.255. El CIDR ayuda a designar direcciones largas con muy pocos caracteres.

En la figura 18 la dirección IP fuente fue establecida para concordar con cualquier ordenador emisor, y la dirección destino fue establecida para concordar con uno en la red de Clase C 192.168.1.0.

Existe un operador que pueda ser aplicado a direcciones IP, el operador de negación. Este le indica a Snort que concuerde con cualquier dirección IP, excepto la indicada. El operador de negación es indicado con un “!”. Por ejemplo, la figura 18 muestra un ejemplo de regla.

```
alert tcp any any -> 192.168.1.0/24 111

(content:"|00 01 86 a5|"; msg:"mountd access";)
```

Figura 18: Ejemplo de Regla de Snort

También se pueden especificar listas de direcciones IP. Una lista de IP's es especificada adjuntando una lista de direcciones IP y bloques CIDR separadas por coma dentro de corchetes. La lista IP no debe incluir espacio entre las direcciones.

```
alert tcp !192.168.1.0/24 any -> 192.168.1.0/24 111 \

     (content: "|00 01 86 a5|"; msg: "external mountd access";)
```

Figura 19: Ejemplo de Regla con negación de dirección IP

Números de Puerto

Los números de puertos pueden ser especificados de varias formas, incluyendo puertos cualquiera, definiciones de puertos estáticos, rangos y por negación.

La forma de especificar cualquier puerto es con la palabra clave "any". Los puertos estáticos son indicados por un número único de puerto como lo es 111 para portmapper, 23 para TELNET o el 80 para http, etc. Un rango de puertos es indicado con el operador de rango ":". Este operador de rango puede ser aplicado de diferentes maneras para tomar diferentes significados, como se ve en la figura 20.

La negación de puerto es indicada haciendo uso del operador de negación ! y puede ser aplicado contra cualquier tipo de regla (excepto any, porque significaría a ninguna). Por ejemplo, si por alguna razón se quería registrar todo, excepto los puertos de la ventana x, se podría hacer algo como en la regla mostrada en la figura 21.

```
Registrar el tráfico udp entrante desde cualquier puerto y
un rango de puertos destino del 1 al 1024.

log udp any any -> 192.168.1.0/24 1:1024

Registrar el tráfico tcp proveniente de cualquier puerto hacia
un puerto menor o igual al 6000

log tcp any any -> 192.168.1.0/24 :6000

Registrar tráfico tcp proveniente de los puertos privilegiados
menores o iguales a 1024 a puertos destinos mayores o iguales
al 500.

log tcp any :1024 -> 192.168.1.0/24 500:
```

Figura 20: Ejemplos de Rangos de Puertos

```
log tcp any any -> 192.168.1.0/24 !6000:6010
```

Figura 21: Ejemplo de negación de puerto

El Operador de Dirección

El operador de dirección -> indica la dirección u orientación del tráfico al que se le aplica la regla. La dirección IP y los números de puerto en el lado izquierdo del operador de dirección son considerados para ser el tráfico entrante del equipo fuente y la información de la dirección y puerto en el lado derecho del operador es la máquina destino. También existe un operador bidireccional <>. Este le indica a Snort que considere la pareja dirección-puerto en cualquiera orientación, fuente o destino. Esto es conveniente para el registro o análisis de ambos lados de la conversación, como lo es en las sesiones de TELNET o POP3. Un ejemplo del uso del uso del operador bidireccional para registrar ambos lados de una sesión TELNET es mostrado en la figura 22. Finalmente, en Snort no existe operador <-.

```
log tcp !192.168.1.0/24 any <> 192.168.1.0/24 23
```

Figura 22: Ejemplo de operador bidireccional

A.3. Opciones de la Reglas

Las opciones de la regla forman parte del corazón de la máquina de detección de Snort. Todas las opciones de la regla están separadas la una de la otra por medio del carácter punto y coma ";". Las palabras claves de las opciones de la regla están separadas por sus argumentos con los dos puntos ":".

Existen cuatro categorías principales de opciones de reglas:

- Meta-data: proveen información acerca de la regla, pero no tienen ningún efecto durante la detección.
- Payload: buscan datos en el interior de la carga útil del paquete y puede ser interrelacionados.
- Non-payload: buscan datos en la otras parte del paquete diferentes a la carga útil.
- Post-detection: Estas opciones son disparadores de reglas específicas que ocurren después que una regla se ha disparado.

Meta-Data Rule Options

msg

La opción *msg* de la regla le indica a la máquina de alerta y registro el mensaje a imprimir junto con el paquete arrojado a pantalla o a una alarma. Esta es una simple cadena de texto que utiliza un "́ como un carácter de escape para indicar un carácter discreto que podría confundir a los analizadores gramaticales de las reglas de Snort (como el carácter punto y coma ";").

Formato

```
msg: "<message text>";
```

reference

La palabra clave *reference* permite a las reglas incluir referencias para sistemas de identificación de ataque externo. Actualmente el plugin soporta varios sistemas específicos igual que URLs único. Este atributo se usa mediante plugins de salida para proveer un enlace con información adicional acerca de la alerta producida (ver tabla 4).

Formato

```
reference: <id system>,<id>; [reference: <id system>,<id>;]
```

sid

Usada excepcionalmente para identificar reglas de Snort. Esta información le permite a los plugins de salida identificar reglas fácilmente. Esta opción debería ser usada con la palabra clave rev.

- <100 reservada para uso futuro
- 100-1.000.000 reglas incluidas con la distribución de Snort
- >1.000.000 usadas para reglas locales

System	URL Prefix
bugtraq	http://www.securityfocus.com/bid/
cve	http://cve.mitre.org/cgi-bin/cvename.cgi?name=
nessus	http://cgi.nessus.org/plugins/dump.php3?id=
arachnids	(currently down) http://www.whitehats.com/info/IDS
mcafee	http://vil.nai.com/vil/dispVirus.asp?virus_k=
url	http://

Tabla 4: Sistemas Soportados

```
alert tcp any any -> any 7070 (msg:"IDS411/dos-realaudio"; \
   flags:AP; content:"|fff4 fffd 06|"; reference:arachnids,IDS411;)


alert tcp any any -> any 21 (msg:"IDS287/ftp-wuftp260-venglin-linux"; \
   flags:AP; content:"|31c031db 31c9b046 cd80 31c031db|"; \
   reference:arachnids,IDS287; reference:bugtraq,1387; \
   reference:cve,CAN-2000-1574;)
```

Figura 23: Ejemplo de uso de referencias

El archivo sid-msg.map contiene un mapa de mensajes de alerta para los IDs de las reglas de Snort.

Formato

```
sid: <snort rules id>;
```

Ejemplo

Este ejemplo es una regla con el ID de Regla Snort de 1000983.

```
alert tcp any any -> any 80 (content:"BOB"; sid:1000983; rev:1;)
```

rev

La palabra clave rev es usada exclusivamente para identificar ediciones de las reglas de Snort. Las ediciones junto con el Id de la regla, permite redefinir y reemplazar patrones y descripciones con información actualizada. Esta opción debe ser usada con la palabra clave sid.

Formato

```
rev: <revision integer>;
```

Ejemplo

Este ejemplo es una regla con número de edición 1.

```
alert tcp any any -> any 80 (content:"BOB"; sid:1000983; rev:1;)
```

classtype

Categoriza las alertas en clases de ataques. Por medio del uso de prioridades. El usuario puede especificar que prioridad tiene cada tipo de clasificación de reglas. Las reglas que tienen una clasificación tendrán una prioridad establecida por defecto.

Formato

```
classtype: <class name>;
```

La clasificación de las reglas está definida en el archivo `classification.config`. La clasificación estándar incluida con Snort está listada en la tabla 5, esta está ordenada actualmente con 3 prioridades por defecto. Una prioridad 1 es el nivel de prioridad mas severo y 4 es el menos severo. Una clasificación de los ataques se puede ver en la tabla 5

Classtype	Description	Priority
attempted-admin	Attempted Administrator Privilege Gain	high
attempted-user	Attempted User Privilege Gain	high
kickass-porn	SCORE! Get the lotion!	high
policy-violation	Potential Corporate Privacy Violation	high
shellcode-detect	Executable code was detected	high
successful-admin	Successful Administrator Privilege Gain	high
successful-user	Successful User Privilege Gain	high
trojan-activity	A Network Trojan was detected	high
unsuccessful-user	Unsuccessful User Privilege Gain	high
web-application-attack	Web Application Attack	high
attempted-dos	Attempted Denial of Service	medium
attempted-recon	Attempted Information Leak	medium
bad-unknown	Potentially Bad Traffic	medium
default-login-attempt	Attempt to login by a default username and password	medium
denial-of-service	Detection of a Denial of Service Attack	medium
misc-attack	Misc Attack	medium
non-standard-protocol	Detection of a non-standard protocol or event	medium

rpc-portmap-decode	Decode of an RPC Query	medium
successful-dos	Denial of Service	medium
successful-recon-largescale	Large Scale Information Leak	medium
successful-recon-limited	Information Leak	medium
suspicious-filename-detect	A suspicious filename was detected	medium
suspicious-login	An attempted login using a suspicious username was detected	medium
system-call-detect	A system call was detected	medium
unusual-client-port-connection	A client was using an unusual port	medium
web-application-activity	Access to a potentially vulnerable web application	medium
icmp-event	Generic ICMP event	low
misc-activity	Misc activity	low
network-scan	Detection of a Network Scan	low
not-suspicious	Not Suspicious Traffic	low
protocol-command-decode	Generic Protocol Command Decode	low
string-detect	A suspicious string was detected	low
unknown	Unknown Traffic	low
tcp-connection	A TCP connection was detected	very low

Tabla 5: Clasificación por defecto de Snort

Advertencia

`Classtype` utiliza clasificaciones definidas por la opción de configuración classification. Las clasificaciones utilizadas por el proveedor de reglas con Snort, son definidas en `etc/classification.config`

priority

La etiqueta de prioridad asigna un nivel de severidad a las reglas. Una regla classtype asigna una prioridad por defecto que puede ser eliminada con el atributo `priority` de la regla.

```
alert tcp any any -> any 80 (msg:"EXPLOIT ntpdx overflow"; \
    dsize: >128; classtype:attempted-admin; priority:10 );

alert tcp any any -> any 25 (msg:"SMTP expn root"; flags:A+; \
   content:"expn root"; nocase; classtype:attempted-recon;)
```

Figura 24: Ejemplo de reglas Classtype

Formato

```
priority: <priority integer>;

alert TCP any any -> any 80 (msg: "WEB-MISC phf attempt"; flags:A+; \
      content: "/cgi-bin/phf"; priority:10;)
```

Figura 25: Ejemplo de prioridad de la regla

Payload Detection Rule Options

content

La palabra clave content es una de las características más importantes de Snort. Esta le permite al usuario establecer reglas que busquen un contenido específico en la carga útil del paquete y disparar respuestas basándose en esos datos. En cualquier momento en el que el patrón de esta opción tenga correspondencia, se realiza la prueba contra los contenidos del paquete. Si los datos concuerdan exactamente y la cadena de datos está contenida en cualquier lugar dentro de la carga útil del paquete, se dice que la prueba es acertada y

se realizan el resto de pruebas de las opciones de la regla. Este atributo puede contener texto mezclado y datos binarios. Los datos binarios son generalmente encerrados dentro del carácter pipe (|) y representado en lenguaje máquina (códigos de byte). El código de byte representa datos binarios como números hexadecimales. La figura 26 contiene un ejemplo de texto combinado y datos binarios en una regla de Snort. Si la regla está precedida por un `!` la alerta será disparada en paquetes que no contengan ese contenido. Esto es útil cuando se quieren escribir reglas que busquen alertar paquetes que no concuerden con ciertos patrones.

Formato

```
content: [!] "<content string>";
```

Ejemplo

```
alert tcp any any -> any 139 (content:"|5c 00|P|00|I|00|P|00|E|00 5c|";)
```

Figura 26: Mezcla de Bytecode Binario y Texto en un 'content'

```
alert tcp any any -> any 80 (content:!"GET";)
```

Figura 27: Ejemplo de Negación

Cambiando el comportamiento de Content La palabra clave content tiene varias palabras claves modificadoras. Las palabras claves modificadoras cambian el trabajo de content previamente especificado. Estos modificadores son:

1. depth
2. offset
3. distance
4. within
5. nocase
6. rawbytes

nocase Permite especificar que Snort debería buscar el patrón específico, ignorando los casos. La palabra clave nocase modifica el content previo en la regla (debe haber un content previo en la regla).

Formato

```
nocase;
```

Ejemplo

```
alert tcp any any -> any 21 (msg:"FTP ROOT"; content:"USER root"; nocase;)
```

Figura 28: Regla de contenido con el modificador nocase

rawbytes Permite a la regla buscar en el paquete de datos original ignorando cualquier decodificación que haya sido hecha por los preprocesadores. Este actúa como modificador de content (debe haber un content previo en la regla).

Formato

```
rawbytes;
```

Ejemplo
Este ejemplo dice cual es el patrón a buscar en el trafico original, en vez de buscarlo en el trafico decodificado proveniente del decodificador TELNET.

```
alert tcp any any -> any 21 (msg: "Telnet NOP"; content: "|FF F1|"; rawbytes;)
```

depth Permite especificar que tan lejos Snort debería buscar dentro de un paquete un patrón especificado. Depht modifica la palabra content definida previamente en la regla.

Un depth de 5 le podría decir a Snort que solo busque el patrón específico dentro de los primeros 5 bytes de la carga útil.

Formato

```
depth: <number>;
```

offset Permite especificar dónde comenzar la búsqueda de un patrón dentro de un paquete. Offset modifica el content previo en la regla (debe haber un content previo en la regla).

Un offset de 5 le diría a Snort que comience a buscar por el patrón especificado después de los 5 primeros bytes de la carga útil.

Formato

```
offset: <number>;
```

El ejemplo de la figura 29 indica que se salte los primeros 4 bytes y se busque cgi-bin/phf en los siguientes 20 bytes.

```
alert tcp any any -> any 80 (content: "cgi-bin/phf"; offset:4; depth:20;)
```

Figura 29: Regla de Content Combinado, Offset y Depth

.

distance Permite especificar hasta dónde Snort debería ignorar el contenido del paquete antes de buscar el patrón especificado relacionado con el final del patrón previamente encontrado.

Se puede pensar que es la misma cosa que depth, excepto que éste está relacionado con el final del último patrón encontrado en vez del principio del paquete.

Formato

```
distance: <byte count>;
```

Ejemplo

La regla mostrada en la figura 30 muestra una expresión de /ABCDE.{1}EFGH/.

```
alert tcp any any -> any any (content:"ABC"; content: "DEF"; distance:1;)
```

Figura 30: Ejemplo de uso de la distancia

within Es un modificador de content que asegura que la mayoría de N bytes están entre los patrones encontrados usando el content. Es usado con la palabra clave distance.

La regla mostrada en la figura 31 obliga a que la búsqueda no pase de 10 bytes después de haber encontrado ABCDE.

Formato

```
within: <byte count>;
```

```
alert tcp any any -> any any (content:"ABC"; content: "EFG"; within:10;)
```

Figura 31: Ejemplo de uso de within

uricontent

Busca las peticiones normalizadas del campo URI. Esto significa que si se está escribiendo reglas que están normalizadas, estas reglas no alertarán. La razón es que las cosas que estás buscando están normalizadas fuera de la memoria URI.

Cuando se escribe una regla uricontent, se escribe el contenido que se quiere encontrar en el contexto en el que el URI será normalizado. Se pueden escribir reglas que busquen el contenido no normalizado usando la opción content.

Formato

```
uricontent:[!]<content string>;
```

Ejemplo

La URI:

```
/scripts/..%c0%af../winnt/system32/cmd.exe?/c+ver
```

será normalizada como:

```
/winnt/system32/cmd.exe?/c+ver
```

Otro ejemplo, la URI:

```
 /cgi-bin/aaaaaaaaaaaaaaaaaaaaaaaaa/..%252fp%68f?
```

será normalizada como:

```
/cgi-bin/phf?
```

isdataat

Verifica que la carga útil tiene datos en un sitio especificado. Opcionalmente busca por datos relativos al final del contenido anteriormente encontrado.

Formato

```
isdataat:<int>[,relative];
```

Ejemplo

```
alert tcp any any -> any 111 (content:"PASS"; isdataat:50,relative; \
   content:!"|0a|"; distance:0;)
```

Esta regla busca la existencia de la cadena PASS en el paquete, luego verifica que hay al menos 50 bytes después del final de la cadena PASS, después verifica que no hay un carácter de nueva linea (Enter) dentro de los 50 bytes del final de la cadena PASS.

pcre

Permite a las reglas ser escritas usando expresiones regulares compatibles con Perl. Para más detalles sobre que se puede hacer mediante expresiones regulares pcre, visitar el sitio web de PCRE `http://www.pcre.org`.

Formato

```
pcre:[!]"(/<regex>/|m<delim><regex><delim>)[ismxAEGRUB]";
```

i	Insensible a mayúsculas
s	Incluye nuevas lineas en el metacaracter punto
m	Por defecto la cadena es tratada como una gran línea de caracteres.^ y $ encuentran al comienzo y final de la cadena. Cuando m es establecida, ^ y $ encuentran seguido o antecedido inmediatamente de cualquier nueva línea en la memoria, también como al principio al final del buffer.
x	Espacios en blanco en el patrón son ignorados excepto cuando evaden o están dentro de una clase carácter.

Tabla 6: Modificadores compatibles con Perl

A	El patrón debe concordar solo al comienzo del buffer (lo mismo que ^)
E	Establece $ para concordar solo al final de la cadena en cuestión. Sin E, $ también encuentra inmediatamente antes del character final si este es una nueva línea (pero no antes de cualquier otra nueva línea).
G	Invierte la “avaricia” de los cuantificadores de tal manera que ellos no ambicionen por defecto, pero se convierte en ambicioso si es seguido por “?”.

Tabla 7: Modificadores compatibles con PCRE

R	Encuentra relative al final del ultimo patrón encontrado. (Similar a distance:0;)
U	Encuentra los buffer del decodificador URI (Similar a `uricontent`)
P	Match normalized HTTP request body (Similar to `uricontent`)
B	No usa los buffer de los decodificadores (Similar a rawbytes)

Tabla 8: Modificadores específicos de Snort

Los modificadores R y B no deben usarse juntos.

Ejemplo

Este ejemplo actúa como una búsqueda no sensible a mayúsculas para la cadena BLAH en la carga útil.

```
alert ip any any -> any any (pcre:"/BLAH/i";)
```

byte test

Prueba un campo de byte contra un valor específico (con operador). Capaz de probar valores binarios o convertir cadenas de bytes representativas a su binario equivalente y probarlos.

Formato

```
byte_test: <bytes to convert>, [!]<operator>, <value>, <offset>  \

    [,relative] [,<endian>] [,<number type>, string];
```

Opción	Descripción
`bytes_to_convert`	Número de bytes a recoger del paquete.
`operator`	Operaciones para comprobar el valor:: ▪ < - menor que ▪ > - mayor que ▪ = - igual ▪ ! - no ▪ & - AND binario ▪ ^ OR binario
`value`	Valor para probar contra el valor convertido
`offset`	Número de bytes en la carga útil para empezar a procesar
`relative`	Usa un desplazamiento relative para el ultimo patron encontrado
`endian`	El tipo endian de número que está siendo leído: ▪ `big` - Procesa los datos como BigEndian (default) ▪ `little` - Procesa los datos como little endian
`string`	El dato es guardado en formato string en el paquete
`number type`	Tipo de numero que está siendo leído: ▪ `hex` - La cadena convertida es representada en Hexadecimal ▪ `dec` - La cadena convertida es representada en Decimal ▪ `oct` - La cadena convertida es representada en octal

Cualquiera de los operadores puede también incluir *!* para verificar si el operador no es verdadero. Si *!* es especificado sin un operador, entonces el operador es establecido a =.

Ejemplo

```
alert udp any any -> any 1235 \

     (byte_test: 3, =, 123, 0, string, dec; \
```

```
msg: "got 123!";)
```

byte jump

Permite escribir las reglas para protocolos codificados de longitud trivial.

Teniendo una opción que lea la longitud de una porción de datos, saltando hacia delante en el paquete, se pueden escribir reglas que salten a una porción específica de la parte del protocolo codificado y hacer detección en muchos lugares específicos.

Formato

```
byte_jump: <bytes_to_convert>, <offset> \

        [,relative] [,multiplier <multiplier value>] [,big] [,little][,string]\

        [,hex] [,dec] [,oct] [,align] [,from_beginning];
```

Ejemplo

```
alert udp any any -> any 32770:34000 (content: "|00 01 86 B8|"; \

              content: "|00 00 00 01|"; distance: 4; within: 4; \

              byte_jump: 4, 12, relative, align; \

              byte_test: 4, >, 900, 20, relative; \

              msg: "statd format string buffer overflow";)
```

Figura 32: Uso de byte jump

Opciones de la regla de Non-payload

fragoffset

La palabra clave fragoffset permite comparar el campo offset del fragmento IP contra un valor decimal. Para atrapar todo el primer fragmento de una sesion IP, se puede usar la palabra fragbits y buscar la mayoria de las opciones de fragmentos en conjunto con un fragoffset de 0.

Formato

```
fragoffset:[<|>]<number>;
```

Ejemplo

```
alert ip any any -> any any \

      (msg: "First Fragment"; fragbits: M; fragoffset: 0;)
```

ttl

La palabra ttl es usada para revisar el valor de tiempo de vida IP. Esta opcion fue propuesta para usarla en la deteccion de intentos de traceroute.

Formato

```
ttl:[[<number>-]><=]<number>;
```

Ejemplo
Este ejemplo revisa que el TTL es menor que 3.

```
ttl:<3;
```

Este ejemplo revisa que el TTL está entre 3 y 5.

```
ttl:3-5;
```

tos

La palabra clave tos es usada en el campo TOS de IP para un valor especifico.

Formato

```
tos:[!]<number>;
```

Ejemplo
Este ejemplo busca un valor de tos que no es 4.

```
tos:!4;
```

id

La palabra id es usada para revisar el campo ID de IP para un valor específico. Algunas herramientas (exploits, escaneadores y otros programas extraños) establecen este campo específicamente para varios propósitos.

Formato

```
id:<number>;
```

Ejemplo
Este ejemplo busca la ID 31337.

```
id:31337;
```

ipopts

Usada para revisar si una opcion IP específica está presente.
Las siguientes opciones pueden ser chequeadas:

rr - Record route (Registro de la ruta seguida)

eol - End of list (Final de la lista)

nop - No Op

ts - Time Stamp (Registra la ruta y los tiempos)

sec - IP security option (Opciones de seguriodad IP)

esec - IP Extended Security

lsrr - Loose source routing (Se dicen routers por los que debe pasar, pero puede pasar también por otros)

ssrr - Strict source routing (Se dice salto a salto la ruta que debe seguir)

satid - Stream identifier (Indentificador de Flujo)

any - any IP options are set (Cualquiera de las opciones IP que estan establecidas)

Las opciones mas frecuentemente vigiladas son strict y loose source routing, las cuales no son usadas ampliamente en las aplicaciones de Internet.

Formato

```
ipopts:<rr|eol|nop|ts|sec|esec|lsrr|ssrr|satid|any>;
```

Ejemplo
Este ejemplo busca la opcion IP de Loose Source Routing.

```
ipopts:lsrr;
```

Advertencia
Solo se puede usar una ipopts por regla.

fragbits

La palabra clave fragbits es usada para revisar si están establecidos en la cabecera IP los bits de fragmentación y bits reservados.

Los siguientes bits pueden ser chequeados:

M - More Fragments

D - Don't Fragment

R - Reserved Bit

Se pueden establecer los siguientes modificadores para cambiar el orden de los criterios de concordancia:

+ compara los bits especificados, mas cualquier otro

- compara si cualquiera de los bits especificados estan establecidos

! compara si el bit especificado no esta establecido

Formato

```
fragbits:[+*!]<[MDR]>;
```

Ejemplo

Este ejemplo inspecciona si el bit de More Fragments (M) y el bit Do No fragment estan establecidos.

```
fragbits:MD+;
```

dsize

Usada para probar el tamaño de la carga útil del paquete. Éste atributo se puede usar para inspeccionar tamaños de paquetes anormales. En muchos casos es útil para detectar buffer overflows.

Formato

```
dsize: [<>]<number>[<><number>];
```

Ejemplo
Este ejemplo busca un dsize entre 300 y 400 bytes.

```
dsize:300<>400;
```

flags

Se usa para revisar si esta presente un bit de bandera TCP.
Los siguientes bits pueden ser chequeados:

F - FIN (LSB in TCP Flags byte) (indica el final de una conexión)

S - SYN (indica el inicio de una conexión)

R - RST (Significa Reset, ha habido algún error y la conexión debe cerrarse)

P - PSH (El segmento contiene datos "Pushed", es decir, entregar datos inmediatamente. Se utiliza para vaciar un buffer de transmisión)

A - ACK (El campo número de acuse de recibo tiene sentido)

U - URG (El segmento contiene datos urgentes)

1 - Reserved bit 1 (MSB in TCP Flags byte)

2 - Reserved bit 2

0 - No TCP Flags Set

Los siguientes modificadores pueden ser establecidos para cambiar el orden de los criterios de concordancia:

+ - compara los bits especificados, más cualquier otro

***** - compara si cualquiera de los bits especificados están establecidos

! - compara si el bit especificado no está establecido

Formato

```
flags:[!|*|+]<FSRPAU120>[,<FSRPAU120>];
```

Ejemplo

Este ejemplo revisa si solo laa banderas SYN y FYn estan establecidas ignorando bit reservado 1 y 2.

```
alert tcp any any -> any any (flags:SF,12;)
```

flow

Este atributo permite a las reglas aplicarse únicamente a ciertas direcciones del flujo de tráfico. También permite aplicar las reglas sólo a clientes o servidores. También permite distinguir a los paquetes relacionados con los clientes de la $HOME_NET que visitan sitios web, de los servidores de la $HOME_NET.

Opción	**Descripción**
`to_client`	Disparador sobre respuestas de servidores de A a B
`to_server`	Disparador sobre peticiones de servidores de A a B
`from_client`	Disparador sobre peticiones de clientes de A a B
`from_server`	Disparador sobre respuestas de servidores de A a B
`established`	Dispara solo Conexiones TCP establecidas
`stateless`	Dispara sin tener en cuenta el estado del preprocesador de flujo (útil para paquetes que han sido diseñados para causar que la máquina se caiga)
`no_stream`	No dispara sobre paquetes de flujo reconstruidos (útil para dsize y stream4)
`only_stream`	Sólo dispara sobre paquetes de flujo reconstruidos

Formato

```
flow: [(established|stateless)]

      [,(to_client|to_server|from_client|from_server)]

      [,(no_stream|only_stream)];
```

Ejemplo

```
alert tcp !$HOME_NET any -> $HOME_NET 21 (msg:"cd incoming detected"; \

    flow:from_client; content:"CWD incoming"; nocase;)

alert tcp !$HOME_NET 0 -> $HOME_NET 0 (msg: "Port 0 TCP traffic"; \

    flow:stateless;)
```

flowbits

Permite a la regla rastrear estados a través de las sesiones del protocolo de transporte. La opción flowbits es las más utilizada para sesiones TCP, ya que permite genéricamente rastrear el estado de un protocolo de aplicación.

Existen siete palabras claves asociadas con flowbits. La mayoría de las opciones necesitan un nombre de usuario definido para el estado específico que está siendo revisado. Esta cadena debería estar limitada por cualquier cadena alfanumérica incluyendo puntos, comas y guión bajo.

Opción	Descripción
`set`	Establece el estado especificado para el flujo actual.
`unset`	No establece el estado especificado para el flujo actual.
`toggle`	Establece el estado especificado si el estado no está establecido y viceversa.
`isset`	Revisa si el estado especificado esta establecido.
`isnotset`	Revisa si el estado especificado no esta establecido.
`noalert`	Hace que la regla no genere alerta, independientemente de las otras opciones de detección.

Formato

```
flowbits: [set|unset|toggle|isset|reset|noalert][,<STATE_NAME>];
```

Ejemplo

```
alert tcp any 143 -> any any (msg:"IMAP login";

  content:"OK LOGIN"; flowbits:set,logged_in;
```

```
  flowbits:noalert;)


alert tcp any any -> any 143 (msg:"IMAP LIST"; content:"LIST";
  flowbits:isset,logged_in;)
```

seq

Usada para revisar un número de secuencia específico TCP.

Formato

```
seq:<number>;
```

Ejemplo
Este ejemplo busca un número de secuencia de 0.

```
seq:0;
```

ack

Usada para verificar un número ack.

Formato

```
ack: <number>;
```

Ejemplo
Este ejemplo busca un ack de 0.

```
ack:0;
```

window

Usada para verificar un tamaño de ventana específico.

Formato

```
window:[!]<number>;
```

Ejemplo
Este ejemplo busca un tamaño de ventana de 55808.

```
window:55808;
```

itype

Sirve para inspeccionar un valor tipo ICMP específico.

Formato

```
itype:[<|>]<number>[<><number>];
```

Ejemplo
Este ejemplo busca un tipo ICMP mayor que 30.

```
itype:>30;
```

icode

Se usa para revisar el valor de un código ICMP específico.

Formato

```
icode: [<|>]<number>[<><number>];
```

Ejemplo
Este ejemplo busca un código ICMP mayor que 30.

```
icode:>30;
```

icmp id

Usada para revisar un valor ICMP ID específico.

Este atributo es útil porque algunos programas de cobertura de canal usan campos ICMP estáticos cuando se comunican. Este atributo en particular fue desarrollado para detectar el ataque de denegación de servicio distribuido "stacheldraht".

Formato

```
icmp_id:<number>;
```

Ejemplo
Este ejemplo busca un ICMP ID de 0.

```
icmp_id:0;
```

icmp seq

Usada para revisar un valor de secuencia ICMP específico. Se creó con el mismo fin que el anterior atributo.

Formato

```
icmp_seq:<number>;
```

Ejemplo
Este ejemplo busca una secuencia ICMP de 0.

```
icmp_seq:0;
```

rpc

Usada para revisar una aplicación RPC, versión y números de procedimientos en peticiones SUNRPC CALL.

Formato

```
rpc: <application number>, [<version number>|*], [<procedure number>|*]>;
```

Ejemplo
El siguiente ejemplo busca una petición RPC portmap GETPORT.

```
alert tcp any any -> any 111 (rpc: 100000,*,3;);
```

ip proto

La palabra clave ip_proto permite chequear contra la cabecera del protocolo IP.

Formato

```
ip_proto:[!|>|<] <name or number>;
```

Ejemplo
Este ejemplo busca tráfico IGMP.

```
alert ip any any -> any any (ip_proto:igmp;)
```

sameip

Permite revisar si la dirección IP fuente es la misma que la del destino.

Formato

```
sameip;
```

Ejemplo
Este ejemplo busca cualquier tráfico donde la dirección IP origen y destino es la misma.

```
alert ip any any -> any any (sampeip;)
```

Post-detección de Reglas

logto

Indica que hay que registrar todos los paquetes que disparan ésta regla en un archivo de registro de salidas especial. Es útil para combinar datos de cosas como actividades nmap, scaneos HTTP CGI, etc. Esta opción no se usa cuando Snort trabaja en modo de registro binario.

Formato

```
logto:"filename";
```

session

Utilizada para extraer los datos de usuarios de sesiones TCP. Hay muchos casos donde ver qué usuarios están tecleando en TELNET, rlogin, FTP, o incluso en sesiones web es muy útil. Existen dos argumentos disponibles para este atributo, printable o all. La palabra clave printable sólo imprime datos que el usuario normalmente ve o es capaz de teclear. Por su lado, all sustituye caracteres no imprimibles con su equivalente hexadecimal.

Formato

```
session: [printable|all];
```

Ejemplo
El siguiente ejemplo registra todas las cadenas imprimibles en un paquete TELNET.

```
log tcp any any <> any 23 (session:printable;)
```

resp

Se usa para intentar cerrar sesiones cuando se dispara una alerta. En Snort esto se llama respuesta flexible. Las respuestas flexibles soportan los siguientes mecanismos de intento de cierre de sesiones:

Opción	Descripción
`rst_snd`	Envía paquetes TCP-RST al socket emisor
`rst_rcv`	Envía paquetes TCP-RST al socket receptor
`rst_all`	Envía paquetes TCP-RST en ambas direcciones
`icmp_net`	Envía un ICMP_NET_UNREACH al emisor
`icmp_host`	Envía un ICMP_HOST_UNREACH al emisor
`icmp_port`	Envía un ICMP_PORT_UNREACH al emisor
`icmp_all`	Envía todo clase de paquetes ICMP al emisor

Estas opciones pueden ser combinadas para enviar múltiples respuestas al host objetivo.

Formato

```
resp: <resp_mechanism>[,<resp_mechanism>[,<resp_mechanism>]];
```

Ejemplo

El siguiente ejemplo intenta resetear cualquier conexión TCP al Puerto 1524.

```
alert tcp any any -> any 1524 (flags:S; resp:rst_all;)
```

react

La palabra clave react, basada en respuestas flexibles, implementa reacciones flexibles al tráfico que concuerda con una regla de Snort. La reacción básica es bloquear sitios interesantes al usuario a los cuales quiere acceder. El código de respuestas flexibles le permite a Snort cerrar conexiones abusivas activamente, y/o enviar una nota visible al navegador. La nota incluye un comentario propio. Los siguientes argumentos (modificadores básicos) son válidos para ésta opción:

- block - Cierra conexiones y envía una nota visible.

Los argumentos básicos pueden ser combinados con los siguientes argumentos (modificadores adicionales):

- msg - incluye el texto de la opcion msg dentro de la nota visible de bloqueo
- proxy <port_nr> - Usa el puerto proxy para enviar la nota visible

Múltiples argumentos adicionales se separan por una coma. La palabra clave react sería situada como la última en la lista de opciones.

Formato

```
react: block[, <react_additional_modifier>];
```

Ejemplo

```
alert tcp any any <> 192.168.1.0/24 80 (content: "bad.htm"; \
    msg: "Not for children!"; react: block, msg, proxy 8000;)
```

tag

Permite a las reglas registrar más paquetes aparte del que disparó la regla. Una vez que una regla es accionada, el tráfico adicional involucrado con la fuente y/o host destino es etiquetado. El tráfico etiquetado es registrado para permitir el análisis de código de respuestas y tráfico post-ataque. Las alertas etiquetadas serán enviadas al mismo plugin de salida como las alertas originales, pero esta responsalbilidad es del plugin de salida para manejar correctamente estas alertas especiales.

Formato

```
tag: <type>, <count>, <metric>, [direction];
```

`type`

- `session` - Registra los paquetes en la session que enciende la regla.
- `host` - Registra los paquetes del host que causo que se activara tag (use el modificador [direction]).

`count`

- `<integer>` - Count es especificado como un número de unidades. Las unidades son especificadas en el campo <metric>.

`metric`

- `packets` - etiqueta el host/sesion para un número <count> de paquetes.
- `seconds` - etiqueta el host/sesion para un número <count> de segundos.
- `bytes` - etiqueta el host/sesion para un número <count> de bytes.

`direction` - Solo es relevante si se usa el tipo de host.

- `src` - La etiqueta del paquete contiene la IP origen del paquete que generó el evento inicial.
- `dst` - La etiqueta del paquete contiene la IP destino del paquete que generó el evento inicial.

Notar, que cualquier paquete que genere una alerta no será etiquetado. Por ejemplo: puede ver que la siguiente regla etiquetará los primeros 600 segundos de cualquier paquete involucrado con 10.1.1.1.

```
alert tcp any any <> 10.1.1.1 any (tag:host,600,seconds,src;)
```

Ejemplo

Este ejemplo registra los 10 primeros segundos de cualquier sesión TELNET:

```
alert tcp any any -> any 23 (flags:s,12; tag:session,10,seconds;)
```

B. Artículo Publicado: "Event-Driven Architecture for Intrusion Detection Systems Based on Patterns"

El trabajo que se ha realizado en este proyecto se ha presentado como parte de *The Second International Conference on Emerging Security Information, Systems and Technologies, (SECURWARE)*. Esta conferencia tuvo lugar en el mes de Agosto del 2008 en Cap Esterel, Francia. En ella se expuso el artículo *Event-Driven Architecture for Intrusion Detection Systems Based on Patterns*.