UNIVERSIDAD CARLOS III DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA TÉCNICA EN
TELECOMUNICACIÓN: TELEMÁTICA

PROYECTO FIN DE CARRERA

DISEÑO E IMPLEMENTACIÓN DE UN MONITOR DE ACCIONES PARA DISPOSITIVOS LIMITADOS

Autora: Cristina Díaz Sierra
Tutora: Florina Almenárez Mendoza
Fecha: 26 Junio 2009

Agradecimientos

A mis padres, a los Telerines y a Miguel:
Por haberme ayudado, comprendido y aguantado.
Gracias

Resumen

Cuando el inventor del primer teléfono móvil, Martin Cooper, realizó la primera llamada telefónica hace treinta años, no imaginó la repercusión y la evolución que su invento podría tener.

Los terminales móviles actuales han evolucionado tanto respecto a sus capacidades de cómputo que nos permiten hacer actividades que hace una década realizábamos con los PCs de sobremesa corrientes: ya no sólo utilizamos los terminales móviles para llamar o mandar mensajes de texto, si no que mandamos correos electrónicos, vemos y modificamos nuestra agenda personal, e incluso, los profesores los utilizan para pasar lista en clase.

El uso extendido de estos dispositivos ha hecho que los cibercriminales tengan como nuevo objetivo los terminales móviles. Por ello, es necesario establecer nuevos sistemas de seguridad orientados a estos dispositivos.

Tras estudiar el comportamiento de los IDS y de los monitores de acciones, se propone un sistema híbrido que monitorice y alerte de todos los posibles ataques que lleguen al dispositivo limitado. Por ello, se diseña e implementa un monitor de acciones prototipo que sea capaz de leer todos los eventos que producen ciertas aplicaciones Java ME instaladas en el sistema y todos los eventos de red producidos y alertar, en caso de que los eventos sean considerados como ataques.

El monitor de acciones se apoyará en un sistema de patrones de ataque y reglas para decidir si el evento que se ha sucedido es un ataque o no.

Todas las reglas y patrones de ataques, además de los eventos que llegan, serán registrados en una base de datos en formato XML que optimizará el limitado espacio en disco del que cuentan estos dispositivos.

Cuando se haya detectado un ataque, el monitor de acciones será capaz de alertar al usuario y registrar el evento en un fichero de log.

Todo el sistema previamente definido se apoya en una interfaz gráfica de usuario, de tal modo que el mismo usuario pueda ver todos los ataques en tiempo real y configurar todos aquellos eventos que quiera definir como ataques.

Abstract

When the inventor of the first mobile phone, Martin Cooper, calls for first time thirty years ago, he couldn't imagine the impact and evolution that his invention was going to have.

Current mobile phones have highly improve their performing operations and we can do with them some of he activities when did ten year ago with normal personal computers. That means that apart from calling or sending text messages, thanks mobile phones, we can send or read emails, read and modify our personal agenda even teachers can call the roll.

As the number of mobile phone has increased, the new aim of hackers is attack mobile phones. That's why it is necessary set up new security systems that could protect mobile phones.

After studying IDS and actions monitor behaviours, we propose a hybrid system that monitors our device and warns us about attacks that happened; taking into account that a new actions monitor prototype is designed and implemented. The actions monitor should be able to detect all the events that some Java ME applications produce. It should also detect all internet events. After detecting the events, the action monitor should be able to decide if those events where attacks or not, and warn in case.

Actions monitor will have an attacks pattern and a rule system that will help it to identify possible attacks among all the events that have happened.

Those rules and potential attacks, as well as events, are registered in a database written in XML format in order to optimize the limited capacity of the phone hard disk.

When actions monitor finds out an attack, it will warn the user and register the event in a log file.

Finally, the application is presented to the user with a simple graphical user interface that will help the user to review the attacks that were found and to configure new rules and potential attacks.

ÍNDICE

Capítulo

1

1 INTRODUCCIÓN

1.1 Motivación

En los últimos años, hemos percibido un notable crecimiento en el número de dispositivos móviles portables (teléfonos móviles, agendas electrónicas, etc.) y, a su vez, un crecimiento en la capacidad de cómputo que éstos nos ofrecen, permitiéndonos cada vez ejecutar nuevas y complejas aplicaciones que complementan su función inicial: accedemos a la red a través de ellos, en sus agendas guardamos todos nuestros datos personales, realizamos compras a través de ellos gracias al sistema *Mobipay…*

Esto hace que el número de ataques y atacantes crezca, de modo que los cíber criminales tienen como nuevo objetivo el ambiente móvil produciendo en los usuarios un posible rechazo hacia el uso de estas nuevas tecnologías. Para evitar esto, es necesario imponer aplicaciones que proporcionen protección de los datos del usuario y del dispositivo móvil.

Existen programas para PC ("*actions monitor*") que nos permiten controlar, mostrar y almacenar todas aquellas actividades que se están produciendo en nuestro sistema mientras que estamos trabajando.

Además, debemos tener en cuenta a los IDS (Intrusion Detection Systems), que son aplicaciones que, además de mostrar y almacenar la información del sistema, son capaces de detectar accesos no autorizados, ya sea nuestro sistema una red o en una computadora, en base a unos patrones y unas reglas definidas.

Teniendo en cuenta todo lo anterior, se propone un prototipo de monitor de acciones que sea sistema híbrido entre un **monitor de acciones** que muestre y controle la información que llegue a nuestro sistema (tanto de red como de aplicaciones) y un **IDS** que analice dicha información en base a unos patrones y reglas establecidas para identificar los posibles ataques que lleguen escondidos dentro de toda esa información.

Este monitor puede ayudar a sistemas de gestión de confianza basados en evidencias para capturar información relevante. Los sistemas de gestión de confianza distribuidos han surgido como un concepto clave para dar soporte a los sistemas de seguridad en entornos dinámicos y abiertos.

Asimismo, un monitor de acciones puede dar la posibilidad al usuario de controlar el uso de recursos, siendo éstos un bien a cuidar en dispositivos limitados, que están consumiendo los procesos para tomar decisiones en determinados contextos.

1.2 Planteamiento del problema

Como ya se ha indicado anteriormente, progresivamente vamos utilizando más los dispositivos limitados para nuestras acciones cotidianas. Además, los usuarios cada vez están más concienciados con el tema de seguridad y pueden rechazar a usar determinadas aplicaciones que puedan producir cualquier riesgo.

Una primera solución que se nos puede ocurrir es la instalación de un *firewall* que nos bloqueé determinadas entradas y/o salidas al sistema actuando como una barrera entre el mundo interior y el exterior o a nuestra aplicación, pero los *firewall* presentan ciertas debilidades, como:

- No pueden incluir la capacidad de combinar varios eventos sucesivos que ocurren con una determinada frecuencia y/o que responden a ciertos comportamientos.
- No todas las amenazas llegan a través del interfaz de red, podemos tener amenazas internas.
- Los *firewall* a su vez también son objeto de ataques.

También debemos contar con que no se encuentran *firewalls o* antivirus de libre distribución para dispositivos limitados, existen pocos y son comerciales. Ejemplos de *firewalls* para PDAs o Pocket PC pueden ser: *"ProtectStar Mobile Firewall"* [18], *"PocketLock for the Pocket PC"* [19] o *"avast! PDA Edition"* [20]. Sería buena idea poder ofrecer una aplicación que sirviera como sistema de seguridad para dispositivos limitados.

Aún así, si el usuario decidiera adquirir un *firewall* o un antivirus y lo instalará en su dispositivo este seguiría desprotegido en el sentido de que no es capaz de proteger al dispositivo durante la ejecución de una aplicación o cuando se está prestando un servicio, por ejemplo, compartir ficheros entre varios dispositivos de la misma red.

Como acabamos de comentar, los ataques también pueden ocurrir desde el interior del dispositivo. Por ejemplo, imaginemos a un usuario autorizado que quiere atacar el sistema o un atacante que finja ser un usuario autorizado. Esto nos dice que los sistemas tradicionales de encriptación, autenticación, autorización e integridad no son suficientes.

Necesitamos *algo* que valga como segunda defensa y además, pueda combinar los tipos de ataques que estamos planteando: los ataques de red y los ataques de aplicaciones software (ataques internos y ataques externos) y a su vez, que nos permita detectar ataques que combinen estos dos tipos de eventos.

1.3 Objetivos

El objetivo de este proyecto es solucionar problemas previamente planteados desarrollando un monitor de acciones prototipo para dispositivos móviles que supervise, almacene y analice toda la actividad generada por una

aplicación software activa en la plataforma Java ME así como la actividad generada a través del interfaz de red del dispositivo que se supervisa. De este modo lo que pretendemos hacer es **proteger** el sistema de ficheros, interfaz de red, aplicaciones críticas, etc.

Se pretende **identificar eventos** con una determinada duración y que ocurran con una determinada frecuencia. Este evento se comparará con unos umbrales previamente definidos en un conjunto de reglas (que pueden ser definidas por el usuario). En función de la comparación anterior se decidirá si dicho evento se considerará como ataque o no.

Si un evento ha sido considerado como ataque, se deberá tomar la acción oportuna que marque la regla que aplica.

También se busca poder definir patrones de ataques para comprobar si un evento que se produzca puede ser un ataque potencial o no.

El monitor de acciones debe presentar una interfaz gráfica que permita al usuario ver todos los eventos que se están produciendo en su dispositivo, configurar el modo de monitorización que desea (pudiendo incluir o no la monitorización del interfaz de red) y configurar las patrones y reglas que se deseen aplicar. Permitiendo añadir patrones y reglas haremos un monitor dinámico que pueda evolucionar al mismo tiempo que la tecnología y, tristemente, los ataques evolucionan.

1.4 Contenido de la memoria

Una vez descrita la motivación y los objetivos que se pretenden cubrir con la realización de este proyecto, a continuación se muestra una pequeña introducción de cada uno de los capítulos que se tratan en esta memoria:

En el **capítulo 2** se expone el estado del arte definiendo y analizando todos los conocimientos adquiridos y que son necesarios para la elaboración del proyecto. De este modo, se estudiarán y clasificarán los detectores de intrusos (IDS), se enseña un detector de intrusos comercial *open source* llamado Snort. Se realiza una pequeña introducción a la seguridad en redes mostrando los ataques más comunes que pueden afectar a los dispositivos móviles. Se presenta la plataforma Java empleada para desarrollar el proyecto: Java ME Personal Profile. Y, por último se estudia el parseador que se emplea para la lectura y escritura de los documentos XML que se utilizarán en la aplicación.

En el **capítulo 3** se describe la estructura de la aplicación realizada, especificando su arquitectura y explicando cada bloque en que divide la aplicación.

En el **capítulo 4** se explica la estructura de clases del monitor de acciones y los diagramas UML que representan dicha estructura de clases. También se expone un diagrama de flujo del hilo principal de la aplicación.

En el **capítulo 5** se presenta la interfaz gráfica del usuario mostrando imágenes de su apariencia y explicando su funcionalidad a modo de guía de usuario.

En el **capítulo 6** se puede ver la batería de pruebas aplicada para cada módulo de la aplicación.

En **capítulo 7** cuenta la historia del proyecto, desarrollando cada fase en la que se ha dividido el proyecto.

En el **capítulo 8** se pueden leer todas las conclusiones sacadas tras realizar el trabajo y las ideas futuras que se tienen para la aplicación.

En el **capítulo 9** aparece la bibliografía del proyecto, dividiendo la misma en libros utilizados, libros electrónicos empleados y recursos consultados en la Web.

El **apéndice A** muestra todos los documentos XML que representan los esquemas (*schemas)* que definen la estructura de los registros de eventos utilizados en la aplicación.

El **apéndice B** contiene los pasos que se han de seguir para poder instalar la aplicación desarrollada y poder empezar a utilizarla

Capítulo

2

2 ESTADO DEL ARTE

En este capítulo se expondrán todos los conocimientos que han sido necesarios para la elaboración de un monitor de acciones para dispositivos limitados utilizando Java ME

Se definirá a un IDS o sistema de detección de Intrusos, que es la tecnología en la que se basará el monitor de acciones para dispositivos limitados. Veremos su clasificación dependiendo de las distintas funciones que pueda tener y explicaremos la arquitectura de los dos estándares que podemos encontrar.

Teniendo en cuenta qué es un IDS, se estudiará el concepto de monitor de acciones y la importancia de monitorizar redes y aplicaciones.

Tras estudiar los sistemas de detección de intrusos se mostrará un ejemplo *open source* de IDS comercial para redes llamado Snort. Snort cuenta con un sistema de reglas bastante poderoso para detectar ataques. Veremos su arquitectura y funcionamiento así como el sistema de reglas.

A continuación, veremos la importancia de la seguridad en redes definiendo los ataques más comunes o intrusos más populares que pueden llegar a cualquier dispositivo limitado.

Para poder desarrollar la aplicación es necesario conocer un lenguaje de programación adecuado, es por ello que también se dedicará una sección a estudiar la tecnología Java Personal Profile utilizada.

Dedicaremos un apartado al estudio de KXML, que es un parseador de documentos XML de libre distribución orientado a dispositivos con baja capacidad de memoria como son los dispositivos limitados.

2.1 IDS (Intrusion Detection System)

2.1.1 Definición

La **detección de intrusos** se puede definir como el proceso de monitorizar actividades dentro de un sistema, el cual puede ser un ordenador o una red por medio de un mecanismo llamado IDS [1] [4].

Un **IDS** almacena la información de la actividad del sistema que monitoriza para posteriormente analizarla y determinar si alguna de las actividades generadas viola un conjunto de reglas previamente establecidas **identificando** así un **ataque**. Tras haber identificado el ataque se generará una alerta indicando que ha sucedido un evento inusual.

A continuación se muestras las **funcionalidades** de los IDS para entender mejor lo que puede y no puede realizar un IDS:

Las funciones generales que un sistema de detección de Intrusos *sí* realiza son [2]:

- Monitoreo y análisis de la actividad de los usuarios y del sistema.
- Auditoría de configuraciones del sistema y vulnerabilidades.
- Evaluación de integridad de archivos de datos y sistemas críticos.
- Reconocimiento de patrones para identificar ataques conocidos.
- Análisis estadístico para patrones con una actividad anormal.
- Reconocimiento de actividades que reflejan violaciones a una política de seguridad.

Las funciones generales que un Sistema de Detección de Intrusos *no* realiza son [2]:

- No puede compensar mecanismos débiles de identificación y autenticación.
- No puede conducir investigación de ataques sin intervención humana.
- No puede compensar las debilidades presentadas por manejo de protocolo de redes.
- No puede compensar los problemas en la calidad o integridad de la información de los sistemas.
- No puede analizar todo el tráfico sobre una red congestionada o utilizada a su capacidad máxima.
- Simplemente informan sobre la llegada de un ataque, pero no indican si dicho ataque ha tenido éxito o no.

2.1.2 Clasificación de los IDS

Podemos clasificar los IDS según sus funciones y su fuente de información. Un IDS puede recopilar eventos desde varios orígenes. Se pueden observar los eventos que llegan a través de la red o parte de la red. También se pueden capturar eventos del sistema o aplicaciones software instaladas en el mismo. Incluso se pueden elaborar IDS combinando varias de las fuentes anteriores.

A continuación mostramos una clasificación de los IDS según funciones y fuentes [1] [2] [3] [4]:

2.1.2.1 IDS basado en red (NIDS, *Network Intrusion Detection System*)

Aquí podremos encontrar la mayor parte de los sistemas de detección de intrusos existentes.

Un IDS basado en red, también llamado **NIDS**, observa los eventos que llegan a través del interfaz de red, analizando y capturando los paquetes que llegan a través del mismo. De este modo, se pueden analizar tanto los eventos del host que tenga instalado este Sistema de Detección de Intrusos como los eventos que producen otros hosts de la misma red.

Ventajas:

Un NIDS colocado estratégicamente dentro de la red puede llegar a monitorizar una red grande.

Un NIDS no afecta al funcionamiento de la red.

Desventajas:

Si estamos en una red grande en la que se genera una gran cantidad de tráfico, el NIDS puede tener dificultades para procesar todos los paquetes de red recibidos.

Los NIDS no pueden analizar el tráfico que llega cifrado.

2.1.2.2 IDS basado en host (HIDS, *Host Intrusion Detection System*)

Un IDS basado en aplicación, conocido como **HIDS**, es un sistema de detección de intrusos que opera sobre la información recogida desde dentro de un host, como por ejemplo, el sistema operativo o sistema de ficheros. Los HIDS estarán instalados dentro del host monitoreado.

Esto permite que se identifiquen los eventos de manera más precisa, reconociendo a los usuarios y procesos que han generado dichos procesos.

Ventajas:

Pueden analizar eventos de origen cifrado si dicho evento se cifra en el host origen y, posteriormente, se descifra dentro del host destino.

Desventajas:

Son más difíciles de administrar, ya que deben ser administrados y configurados dentro de cada host monitoreado.

Si un ataque **DoS** llega a la máquina, el IDS puede ser deshabilitado, ya que estos ataques pueden usar todos los recursos del host, influyendo en el rendimiento del equipo.

NOTA: Los dos tipos de IDS que aparecen a continuación no son como tales tipos de IDS pero sí los podemos encontrar así en algunas literaturas [3]:

2.1.2.3 IDS basado en aplicación

Un IDS basado en aplicación coge la información a nivel de aplicación a partir de los logs que generan las mismas.

Ventajas:

Se pueden monitorizar con precisión aplicaciones específicas.

Desventajas

No es posible identificar la integridad de la aplicación base con relación al monitoreo y estrategias de detección.

2.1.2.4 IDS basado en objetivos

Un IDS basado en objetivos monitoriza archivos, aplicaciones específicas, cambios de atributos, cambios en el sistema de ficheros, etc. en función de los ataques objetivo que se quieran descubrir.

Ventajas:

Detección específica a gusto del administrador.

Desventajas:

No distingue entre error humano y atentado contra el sistema

2.1.3 Arquitecturas estándar de IDS

En los siguientes apartados se muestran dos propuestas de arquitectura estándar para los IDS por parte de dos grupos de trabajo CIDF y IDWG [2].

2.1.3.1CIDF (“Common Intrusion Detection Framework”)

El “Common Intrusion Detection Framework” (CIDF) fue desarrollado por el CIDF Working Group en 1997 con el fin de convertir el CIDF en un conjunto de especificaciones que cumplieran [5]:

- Sistemas de Detección de Intrusos ínter operables y con el máximo de información compartida.
- Los componentes del sistema pueden ser reutilizables para otros contextos distintos para los que fueron diseñados.

ARQUITECTURA

Dentro de la arquitectura CIDF podemos definir los siguientes componentes interrelacionados entre sí:

- Generadores de eventos
- Analizadores
- Bases de datos
- Utilidades de respuesta

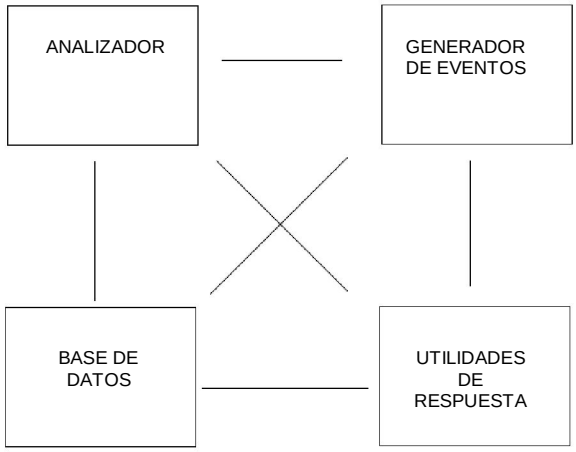


Figura 1: Arquitectura CIDF

En la figura 1 aparece la arquitectura CIDF, mostrando sus componentes y relación entre ellos.

Cada componente es una unidad lógica que se comunica con todos las demás. Esta comunicación se realiza a través de **GIDO** ("Generalized Intrussion Detection Objects"), que es una especificación de mensajería que permite la descripción y consulta de eventos, dirección de acciones y descripción de componentes.

Veamos una descripción más detallada de cada componente:

- **Generadores de Eventos**

El papel principal del generador de eventos es obtener eventos en un ambiente externo al IDS y proporcionarlos, en forma GIDO, al resto del sistema.

Los generadores de eventos proporcionan los eventos al resto del sistema tan pronto ocurren, tratando de tener la mínima demora en el transporte. El almacenamiento de los eventos es llevado a cabo en una base de datos (registro) específico para eventos.

- **Analizadores de Eventos**

Obtienen los eventos (codificados en GIDO) de otros componentes, los analizan y retornan nuevos eventos. En este componente se encuentra el corazón del *algoritmo de detección,* algoritmo que hará las operaciones oportunas para identificar los posibles ataques al sistema.

- **Base de Datos – Eventos**

Representa la funcionabilidad de almacenamiento de un IDS. Almacenan todos los eventos que son encontrados por el generador de eventos.

- **Unidades de Respuesta**

Es el componente que ayuda al IDS a tomar una acción tras la llegada de un evento. También pueden llegar a ejecutar dicha acción (terminar procesos, reiniciar conexiones, alterar permisos de archivos, etc.).

2.1.3.2 IDWG ("Intrusion Detection Working Group")

Al igual que CIDF, el "Intrusion Detection Working Group" define su propio esquema de mensajería y comunicaciones, con el propósito de buscar un estándar en el campo de los IDS [6].

Esta propuesta de mensajería está definida basándose en la utilización de "Extensible Markup Language" (XML).

Mensajería

La ventaja de utilizar XML es que cada documento XML tiene asociado un DTD que define qué estructura debe tener dicho documento. Además los XML pueden ser interpretados utilizando distintos *parser* de libre distribución.

Por otro lado, gracias a la utilización de XML para la representación de un modelo de datos, se puede aplicar cualquier diseño de mensajería que sea **Orientado a Objetos**. Este diseño basado en Orientación a Objetos nos permitirá extender y ampliar el formato de los mensajes según la necesidad particular de cada IDS.

Arquitectura

IDWG no define un esquema general de arquitectura, si no que define un esquema de mensajería y comunicaciones, de modo que cada IDS pueda definir su propia arquitectura adatando así las necesidades de cada sistema o cada caso.

Al igual que para CIDF se distinguen varios módulos o componentes, no necesariamente separados:

- Analizador
- Sensor
- Fuente de datos
- Gestor.

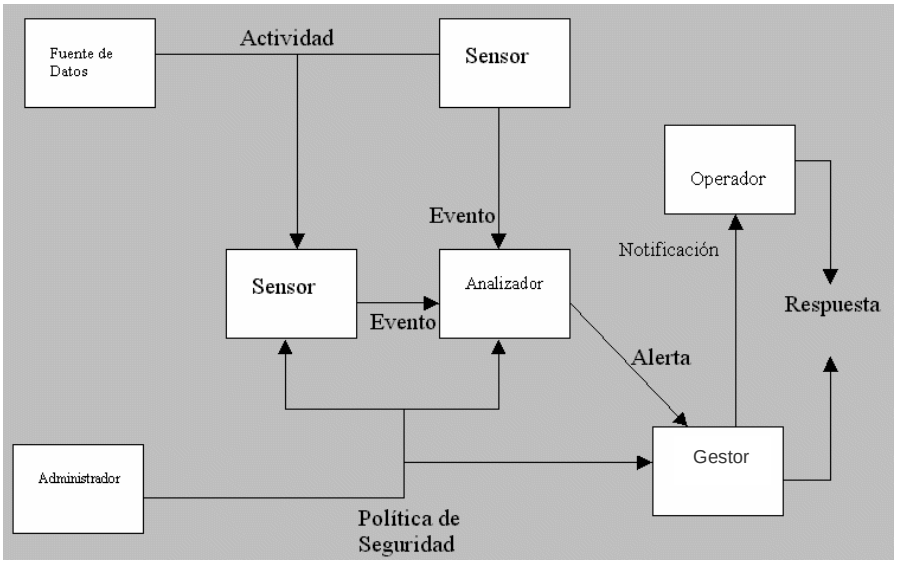


Figura 2: Ejemplo de Arquitectura IDWG

En la figura 2 se muestra la arquitectura de un IDS basada en el estándar IDWG. No todos los IDSs tienen todos los módulos que aquí se muestran, algunos podrían tener más de uno y otros podían tener varios módulos fusionados en uno. A continuación se muestra con más detalle la descripción de los elementos más importantes de los que se compone:

- **Fuente de Datos**

Información externa al IDS que sirve para identificar una actividad no deseada (un ataque potencial). Podemos tener distintas fuentes de datos como el interfaz de red, un log de una aplicación o un log del sistema operativo.

- **Sensor**

Componente del detector de intrusos que recolecta los eventos de la fuente de datos. Envía estos datos al analizador.

- **Analizador**

Es el componente que analiza los datos que son recolectados por el sensor, buscando indicios de que el evento que ha llegado pueda ser una actividad no deseada o no autorizada.

- **Gestor**

Componente o proceso desde el cual se administran los diversos componentes del IDS. Entre sus funciones se incluyen: configurar los sensores y los analizadores, administrar la notificación de eventos, consolidar los datos y generar reportes.

2.1.4 Snort

Snort es un estándar *open source* de detección de intrusos en redes [7].

Es multiplataforma y es útil para monitorizar pequeñas redes TCP/IP. Snort analiza el tráfico en tiempo real, descodificando todos los paquetes que llegan a la red para identificar posibles ataques con la ayuda de patrones y reglas previamente establecidas. Tras detectar un ataque, generará una alerta y añadirá ese evento a un fichero de log. Adicionalmente se podrán bloquear los paquetes maliciosos.

Se puede configurar Snort para que actúe de los tres modos siguientes:

- ***Packet Sniffer***: en este caso Snort actúa como la aplicación *tcpdump*, recibiendo y mostrando todos los paquetes que llegan a través del interfaz de red elegido por el usuario.
- ***Logger de paquetes***: loguea los paquetes recibidos en disco dentro de un fichero previamente configurado. Esto es útil para depurador de tráfico de red.
- ***Detección y prevención de intrusos***: Snort actuará como un IDS, monitorizando todas las fuentes de información para identificar posibles eventos maliciosos o ataques comparando el tráfico recibido con patrones de ataques.

Para poder detectar los paquetes de red, Snort utilizará las librerías de *sniffing* de paquetes promiscua de distribución gratuita "*libcap*" (si estamos ante plataforma Linux) y "*winpcap*" (si nos encontramos en Windows).

Snort basa su diseño en la utilización de plug-ins de modo que permite obtener una flexibilidad ilimitada desde que se pueden añadir o eliminar tantos módulos como se deseen. Se pueden utilizar tanto los módulos que son proporcionados con la distribución de Snort, como módulos creados por el propio usuario.

2.1.4.1 Arquitectura de Snort

Como se acaba de comentar anteriormente, Snort está compuesto de distintos módulos o plug-ins. Veamos a continuación cómo se relacionan unos con otros:

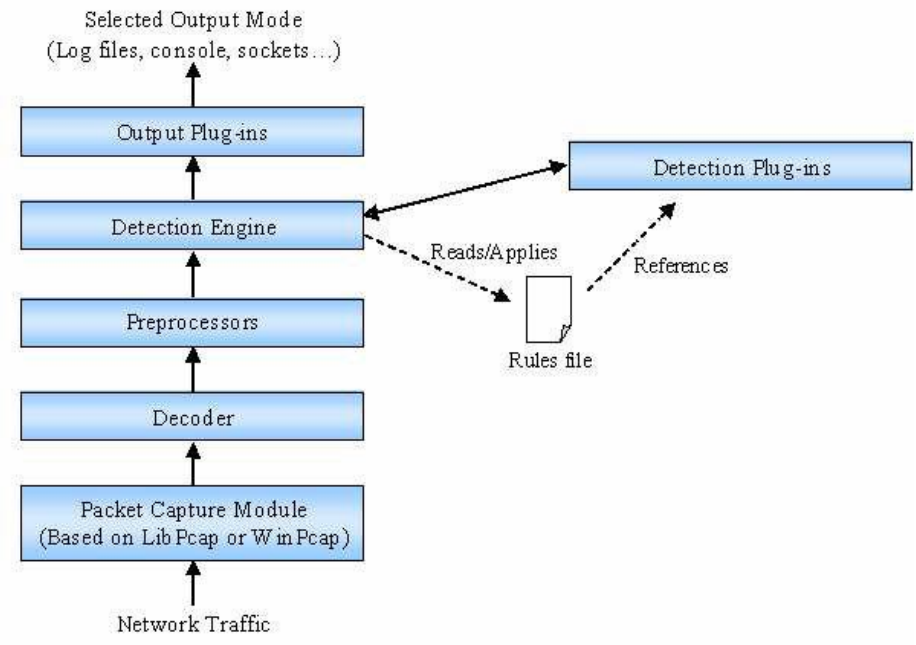


Figura 3: Diagrama de Bloques de Snort

En la figura 3 encontramos el diagrama básico de bloques de Snort en el que se muestran los módulos y plug-ins básicos que cualquier usuario debería tener instalado para conseguir el funcionamiento básico de Snort.

Funcionamiento básico de Snort:

El tráfico de red llega al host por cualquiera de los interfaces de red activos en el dispositivo y es detectado gracias a un capturador de paquetes (**módulo Packet Capture**) que está basado en las librerías de libre distribución LibPcap o WinPcap (dependiendo del sistema operativo frente al que nos encontremos).

Una vez detectado cada paquete, se realizarán las operaciones oportunas dentro del motor principal de Snort para saber si ha llegado un ataque o no. Si se ha encontrado un ataque, Snort accionará una salida (**módulo Output Plug-Ins**). La salida puede consistir en una alerta, la emisión de un mensaje al usuario o el bloqueo de ese paquete, por ejemplo.

Como se ha comentado anteriormente, en el motor principal de Snort se realizan todas las operaciones adecuadas para saber si, ante un paquete determinado, hemos recibido un ataque. Este motor principal o núcleo de la aplicación está compuesto por tres subsistemas primarios: el decodificador de paquetes (**Decoder y Preproccesor**), el motor de detección (**Detection Engine**) y el sistema de alertas y loggings (**módulo Output Plug-Ins**).

Veamos con más detalle cada módulo:

- **Decodificador de paquetes:**

La decodificación de paquetes es el primer paso por el que pasa un paquete en Snort.

El decodificador se encarga de desglosar cada campo de un paquete (cabecera y campo de datos) para identificar qué protocolos intervienen en dicho paquete (Ethernet, ip, icmp, tcp, udp, etc.) Dicha información se guardará junto con el campo de datos del paquete para posteriormente identificar si tenemos o no un ataque.

Para poder guardar la información anterior, este módulo se encargará de parsear cada paquete de red. Al realizar esta operación, Snort tendrá también la capacidad de descubrir malformaciones o anomalías en el tráfico de red. Es decir, podrá encontrar errores en los campos de las cabeceras. Por ejemplo, imaginemos que el decodificador de paquetes de Snort ha encontrado un paquete Ethernet que indica que en su campo de datos se encuentra un paquete IP. Sabemos que el tamaño mínimo de una cabecera IP es de 20 bytes, si el decodificador descubre que la cabecera del paquete que se encuentra dentro del paquete Ethernet recibido es menor que 20, Snort detecta una anomalía y alertará sobre la llegada de dicho paquete. Este paquete se descartará no realizando ninguna operación más con él.

- **Motor de Detección:**

Aquí llegan los paquetes de red decodificados para poder operar con ellos y descubrir si dichos eventos son ataques o no. Para ello, Snort cuenta con un motor de reglas bastante potente.

Cada regla está formada por una cabecera más unas opciones. En las reglas se especifican los eventos que se consideran como ataques y la acción a realizar en caso de que el evento se produzca. Así que cuando un paquete llega al motor de detección se busca entre todas las reglas aquella que case con el evento.

  - Si la regla coincide, se aplicará la acción oportuna que se indique dentro de la regla y se deja de buscar. La acción a realizar será aplicada dentro del sistema de alertas y loggings.
  - Si la regla no coincide, se sigue buscando entre todas las reglas restantes.
  - Si ninguna regla coincide, el evento no se considerará como ataque.

En el apartado 2.3.1.2 explicaremos más en profundidad los detalles de este motor de reglas.

- **Sistema de Alertas y Loggings:**

El sistema de alertas y loggings salta cuado un determinado evento ha violado alguna regla, es decir, cuando se ha detectado un ataque. En este caso se

aplicará la acción que manda la regla. Ejemplos de acciones incluyen una alerta al usuario o el logging del paquete en el fichero de log de Snort.

Este sistema es considerado como la salida de Snort.

2.1.4.2 Sistema de reglas de Snort

Para detectar los posibles ataques que llegan al sistema global, Snort cuenta con un potente motor de reglas.

Podemos definir una **regla** como una instrucción o un conjunto de instrucciones a realizar cuando un evento cumple unas determinadas condiciones indicadas en ella.

Las reglas se componen de dos partes: **Rule Header** (cabecera) y **Rule Options** (opciones):

- **Rule Header**: Contiene todos los atributos comunes a todas las reglas. Estos atributos son: *acción* a realizar si la regla casa, *protocolo* que genera el paquete, *direcciones IP* fuente y destino, *puertos* origen y destino y *máscara de red.*

- **Rule Options:** Contiene los atributos específicos para cada regla, como pueden ser los atributos específicos de cada protocolo, por ejemplo el campo flags del paquete tcp. Además incluye los mensajes de alerta que se mostrarán en caso de que la regla sea aplicada.
  Tenemos cuatro categorías de Rule Options. Cada opción irá separada dentro de la regla por ";":

  - *General options:* opciones que muestran información sobre la regla.
  - *Payload options:* opciones que se refieren al campo de datos del paquete.
  - *Non-payload options:* opciones que se refieren a campos distintos del campo de datos del paquete.
  - *Post-detection:* opciones nos indican atributos que nos servirán en caso de que una regla se dispare.

Las reglas en Snort se mantienen dentro de una lista encadenada bidimensional de "Encabezados de Regla" (Rule Header) y "Opciones de Regla" (Rule Options). A continuación en la figura 4 mostramos un ejemplo de la estructura lógica de las reglas:

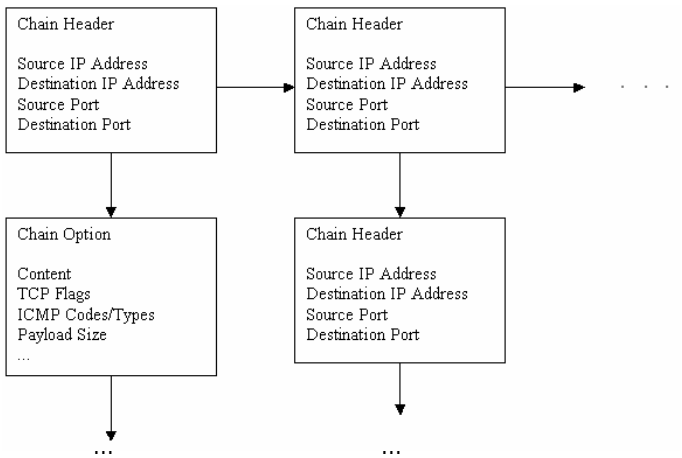


Figura 4: Reglas de Snort

Cuando un evento llega, se busca entre la cadena de reglas de forma recursiva en ambas direcciones. La primera regla que case, será la regla aplicada.

A continuación mostramos un par de ejemplos de reglas empezando por la más sencilla:

```
alert ICMP $EXTERNAL_NET any → $HOME_NET any (msg:"ICMP ping";
dsize:32; sid:123123123123; itype:8; content:
"abcdefghijklmnopqrstuvwxyzabcdefghi"; depth:32)
```

En el ejemplo anterior se muestra una regla que se disparará cuando llegue un ping con destino a nuestro host: se alertará cuando llegue un paquete ICMP con dirección destino cualquier dirección IP de cualquier externa, desde cualquier puerto, con dirección IP destino mi host local a cualquier puerto. Se alertará con el mensaje "ICMP ping".

```
alert tcp any any → 192.168.1.0/24 111 (Content:"100 0186a5"; msg:
"mountd access")
```

En este ejemplo se muestra una regla que alerta en caso de que se intente ejecutar mountd en la red ip 192.168.1.0/24. Se alertará en caso de que un host mediante el protocolo tcp desde cualquier dirección ip y desde cualquier puerto intente acceder al puerto 111 de la red 192.168.1.0/24. Se alertará con el mensaje "mountd access".

Snort incluye un conjunto de reglas genéricas para que el usuario que instala la aplicación tenga unas reglas iniciales básicas. Sin embargo, Snort cuenta con más herramientas para que el usuario pueda añadir más reglas según sus necesidades o pueda actualizar las reglas que tiene instaladas:

- Una primera opción es escribir las reglas según su formato en el fichero de configuración de Snort *snort.conf*
- Se pueden definir nuevas reglas y nuevas firmas de ataques mediante un sistema de *scripting* que contiene Snort.
- Se pueden descargar las últimas actualizaciones de reglas de la página oficial de Snort: www.snort.org

2.2 Monitor de acciones

Definiremos "**monitor de acciones**" como un programa que corre en nuestro sistema controlando, almacenando y mostrando toda actividad que ocurre en nuestro sistema [8].

El programa muestra los procesos lanzados y guarda toda la información procesada en un fichero de texto (un log) para un posterior análisis.

Un monitor de acciones puede realizar un seguimiento de todos los programas y aplicaciones que se abren y se cierran, los registra en el fichero de log comentado anteriormente, y si el usuario lo desea, presenta el informe para así poder comprobar el uso que se hace del ordenador [24]

Esta herramienta es muy útil para ordenadores que son compartidos, de este modo cualquier usuario puede ver el uso que se está realizando del sistema: qué programas se han abierto, quién ha sido la última persona que ha utilizado el equipo, qué procesos se han abierto, etc.

Como ya se ha adelantado anteriormente, un monitor de acciones nos puede dar mucha clase de información: aplicaciones que se están ejecutando actualmente en el equipo, procesos que están abiertos y ejecutándose en el dispositivo y rendimiento del sistema (uso de la CPU, uso de la memoria interna, consumo de energía, etc.)

También cabe descartar que hay determinados monitores de acciones que permiten al usuario terminar algunos procesos o cerrar algunas aplicaciones que se estén ejecutando.

Ejemplos de monitores de acciones pueden ser el "Administrador de Tareas" de Windows obtenido al pulsar las teclas "Ctrl+Alt+Supr" o el monitor de acciones comercial de libre distribución "Actions Monitor 1.02" [8].

2.2.1 Diferencias entre un IDS y un monitor de acciones

A partir de la información proporcionada sobre los IDSs y los monitores de acciones comerciales se pueden extraer las siguientes diferencias:

- Los IDSs son sistemas diseñados para detectar anomalías en función de patrones mientras que los monitores de acciones están pensados para registrar la actividad del sistema.

- Los monitores de acciones actúan sobre un mismo ordenador mientras que los IDSs pueden ser de varios tipos incluyendo la monitorización de redes.

- En los IDSs los usuarios pueden configurar patrones y reglas de los eventos a monitorizar.

- Hay IDS que son capaces de elaborar respuestas frente a eventos, mientras que los monitores de acciones sólo informan.

2.3 Seguridad en redes con dispositivos limitados

Desde el principio de su existencia las redes de ordenadores tenían su uso limitado para el envío de correo electrónico o para compartir recursos, generalmente impresoras.

Hoy en día las redes de ordenadores nos ofrecen mucha más posibilidades de uso: podemos compartir un sistema de ficheros, podemos realizar transferencias bancarias, incluso podemos acceder a otros equipos en remoto haciendo operaciones como si estuviéramos en nuestro equipo local. Esto hace que sea necesario establecer un sistema de seguridad de modo que nos permita realizar todas estas operaciones con confianza evitando el riesgo que se crea al tener una red a disposición de millones de usuarios.

Pero la evolución tecnológica no sólo ha hecho aumentar la capacidad de las redes de ordenadores iniciales, si no que ha creado nuevos tipos de redes. Por ejemplo cada vez el uso de redes inalámbricas está más difundido dentro del ámbito empresarial y el personal.

También es importante destacar que los dispositivos inalámbricos limitados tales como PDAs o móviles cada vez tienen mayor capacidad de cómputo permitiendo realizar operaciones que hace una década hacíamos con un PC normal de sobremesa. Teniendo en cuenta lo anterior, si conectamos uno de estos dispositivos a una red inalámbrica aumentaremos el número de acciones a realizar desde estos dispositivos, es decir, podemos realizar operaciones en nuestra cuenta bancaria desde nuestro PC de sobremesa conectado a una red de área local o desde nuestra PDA conectada a una red inalámbrica.

La proliferación del uso de estos dispositivos limitados dentro de redes inalámbricas ha hecho que los cibercriminales tengan como nuevo objetivo el ambiente móvil. Pero, ¿por qué desearían atacar a los terminales móviles? Veamos los factores principales que hacen que los cibercriminales tengan este nuevo objetivo:

- **Mayor población de dispositivos objetivo:** Como hemos comentado anteriormente en los últimos años ha habido un notable aumento en el número de dispositivos móviles. En el 2009 se espera un aumento del número de móviles vendidos gracias a nuevos terminales especializados en música y fotografía. Con lo cual, los criminales ven a los dispositivos móviles como un objetivo en constante crecimiento.

- **Popularidad con los desarrolladores:** es más probable que los cibercriminales intenten estudiar las vulnerabilidades de los dispositivos más populares.

- **Aumento del poder de cómputo:** los dispositivos móviles actuales tienen instalado software altamente avanzado.

- **Almacenamiento de datos:** los usuarios almacenan todo tipo de datos en estos dispositivos como por ejemplo, mensajes privados SMS o de correo electrónico, agenda personal, números y direcciones de contactos, etc.

- **Capacidades limitadas:** los dispositivos limitados, tal y como su nombre indica, son limitados y no tienen la capacidad de cómputo como la que puede tener cualquier PC de sobremesa, por ello tienen grandes vulnerabilidades y eso facilita la tarea a los atacantes.

Para poder evitar todos los ataques que nos puedan llegar a nuestra red (ya sea red de área local o red inalámbrica), debemos elaborar un sistema de seguridad que nos sirva tanto para redes de área local como para redes inalámbricas y que nos proteja de todos los posibles ataques que podamos tener.

2.3.1 Ataques comunes:

Un buen sistema de seguridad debería protegernos de los ataques más comunes de la red.

A continuación, vemos los ataques más destacados que nos podemos encontrar en redes de área local y en redes inalámbricas con cualquier tipo de dispositivo o host conectado a ella [9] [10]:

Apache Web Server DoS Attack

Este ataque aprovecha la vulnerabilidad que presenta Apache Server en la transferencia de datos por secciones en sus versiones 1.2.x a la 2.0.36.

Apache Web Server contiene un defecto que permite a un atacante remoto ejecutar un código arbitrario. Esto se basa en el mecanismo de poder calcular el tamaño de dato codificado y transferido. El atacante intentará sustituir este trozo de datos por su código malicioso.

En la mayoría de los casos, la consecuencia de este ataque es que el proceso hijo que está corriendo con la petición termine. Esto ayuda al atacante remoto a enviar un ataque de negación de servicio aprovechándose de que el proceso padre está intentando sustituir al proceso hijo que se había terminado.

En una plataforma en la que Apache corra muchos procesos hijos el tiempo que se aplicado a sustituir a todos ellos es muy significante y puede llegar a producirse un gran ataque de negación del servicio.

Smurf Attack

El ataque smurf es un ataque de negación del servicio que utiliza mensajes de ping a una dirección broadcast con objetivo de inundar el sistema que pretende atacar [11].

El atacante manda grandes cantidades de mensajes ICMP (ping) con dirección origen la dirección de la máquina que pretende ser atacada. Todos estos paquetes ICMP se mandarán a la dirección broadcast. Cuando este mensaje ICMP llegue a cualquier máquina que esté escuchando en esa dirección broadcast, ésta mandará un echo reply contestando a dicha petición a la dirección origen del mensaje ping, que es la dirección de la víctima. Como consecuencia de ello, la víctima se verá inundada con múltiples paquetes.

Nmap (TCP) y Nmap (UDP)

Nmap es un escaneador de puertos de libre distribución diseñado para escanear grandes redes, aunque también trabaja en single hosts.

Un escaneador de puertos es un programa que intenta conectarse a una lista o rango de puertos TCP o UDP en una lista de direcciones IP. Nmap usas varios paquetes IP en varias maneras para determinar qué hosts están disponibles en la red, qué servicios (nombre de la aplicación y versión) se están ofreciendo, qué forma tiene el sistema de operar (sistema operativo y versión), qué tipos de paquetes (filtros y firewalls) están en uso y más características.

Para el TCP nmap, el atacante manda una combinación inusual de opciones TCP para ver cómo responde el sistema. Usualmente, el atacante intenta identificar el sistema operativo de la víctima. Esta información puede ayudar al atacante a determinar qué debilidades existen en el sistema y provee información evaluable para realizar ataques más amplios.

Nmap es difícilmente detectable, ha sido creado para evadir los Sistemas de Detección de Intrusos (IDS) e interfiere lo menos posible con las operaciones normales de las redes y de las computadoras que son analizadas.

SYN Flood (TCP)

Este ataque se basa en los tres pasos necesarios y obligatorios para el establecimiento de una conexión TCP. En la figura 5 podemos observar las tres tramas que son necesarias para que una conexión TCP se establezca correctamente:

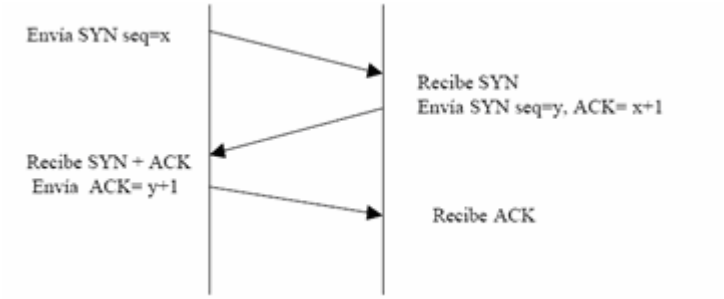


Figura 5: Establecimiento de una conexión TCP.

Cuando un dispositivo móvil recibe el ataque SYN Flooding, recibe un paquete SYN con dirección IP de fuente incorrecta. Cuando el dispositivo intenta responder enviando un paquete de ACK a la falsa IP, tendrá un puerto ocupado durante unos minutos. EL atacante seguirá mandando esos paquetes SYN para ir ocupando los puertos del dispositivo.

El atacado nunca llegará a recibir el paquete confirmación del atacante según se muestra en la figura 5, y se quedará esperando manteniendo de ese modo el puerto ocupado.

Se considera ataque cuando se han enviado 8 paquetes tcp incorrectos en 8 minutos.

UDP Flood (UDP)

Es otro tipo de ataque de negación del servicio en el que no se intenta robar información sino que se intenta deshabilitar la red o el ordenador.

UDP Flood consiste en enviar una gran cantidad de paquetes a un receptor, de tal modo que se desbordará su buffer de recepción y se perderá la conexión con la red.

Ping Flood (ICMP)

Ping flood es un flujo de tráfico ICMP, u otro tipo de denegación de servicio, que consiste en enviar paquetes ICMP suficientemente grandes a la máquina.

La longitud máxima de un datagrama IP es de 64K (65535 Bytes) incluyendo la cabecera del paquete (20 Bytes).

Para enviar un mensaje ICMP tenemos disponibles 65535(datos) 20 (cabecera IP) 8 (cabecera ICMP) = 65507 Bytes.

En el caso de enviar órdenes al sistema operativo para que envie un datagrama de longitud de 65510 bytes (inferior a los 65535) con lo que los datos a enviar cogen un único paquete IP (fragmentado en N trozos, pero pertenecientes al mismo datagrama IP).

Si sumamos: 65510 + 20 + 8 =65538

Debido a que el espacio disponible tan sólo es de 65535 bytes al reensamblar el paquete en el destino se suelen producir errores de overflow/coredump que causan la parada del servicio o del sistema atacado.

2.4 Plataforma Java ME

Dependiendo del sector en el que nos encontremos, Sun en 1999 estructura la tecnología Java 2 (tecnología Java existente en 1999) en las tres plataformas siguientes [12] [13]:

- **Java Enterprise Edition (Java EE):** esta plataforma nos ofrece la posibilidad de desarrollar soluciones orientadas a entornos empresariales como e-commerce o e-business.
- **Java Standard Edition (Java SE):** proporciona un entorno para el desarrollo de aplicaciones para ordenadores de sobremesa y servidores.
- **Java Micro Edition (Java ME):** nos permitirá desarrollar aplicaciones para dispositivos con menor capacidad de procesamiento y menor tamaño de memoria como teléfonos móviles, PDAs, buscas, electrodomésticos inteligentes, etc.

2.4.1 Arquitectura de Java ME

Java ME se estructura en 3 capas para conseguir y proporcionar flexibilidad y adaptación. Veamos cuáles son estas tres capas [12] [13] [14]:

- **Máquina virtual:** Java ME dispone de varias máquinas virtuales Java con diferentes requisitos, cada una para los distintos tipos de pequeños dispositivos.
- **Configuración:** es un conjunto de clases básicas que forman el corazón de las implementaciones para los distintos tipos de dispositivos. Existen dos configuraciones Java ME que veremos con detalle más adelante: CDC y CLDC.
- **Perfiles:** bibliotecas Java con clases específicas para complementar a las configuraciones y añadir funcionalidad dentro de una misma familia de dispositivos. Un dispositivo podrá soportar varios perfiles.

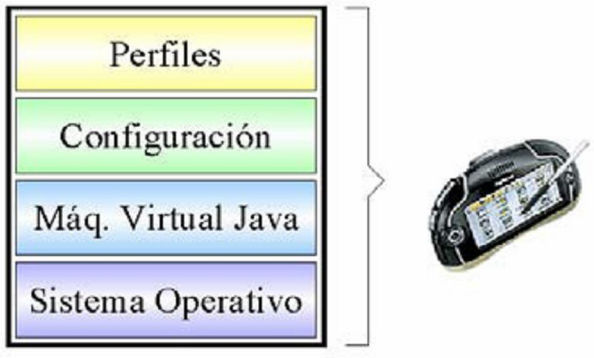


Figura 6: Arquitectura General de Java ME

2.4.1.1 Máquina virtual

Una máquina virtual de Java (**JVM**) es un programa interpreta el código intermedio (bytecode) de los programas Java precompilados a código máquina ejecutable por la plataforma, efectuar las llamadas pertinentes al sistema operativo subyacente y observar las reglas de seguridad y corrección de código definidas para Java. De este modo se proporciona independencia de la plataforma y del sistema operativo subyacente.

Como se ha indicado anteriormente, Java ME consta de dos configuraciones: CDC y CLDC. Como consecuencia de ello, serán necesarias dos máquinas virtuales: **CVM** (**Compact Virtual Machine**), que es la máquina virtual ligada a CDC y **KVM** (**Kilo Virtual Machine**), máquina virtual para CLDC)

2.4.1.2 Configuraciones. CDC y CLDC

Una **configuración** es el mínimo conjunto de APIs para un grupo de dispositivos para así conseguir la funcionalidad básica. Podemos decir que una configuración es una librería básica.

Dentro de las configuraciones se define lo siguiente:

- Se definen las características que se soportan del leguaje de programación Java.
- Se especifican las características que soporta la máquina virtual de Java.
- Se definen las bibliotecas básicas de Java y los APIs que son soportados en la configuración.

Actualmente existen dos configuraciones:

- **Connected Device Configuration (CDC)**
- **Connected Limited Device Configuration (CLDC)**

Connected Device Configuration (CDC)

Esta configuración está orientada a dispositivos con cierta capacidad computacional y de memoria. Por ejemplo, *Internet Screen Phones* o sistemas telemáticos para automóviles.

CDC está enfocada a dispositivos con las siguientes capacidades:

- 512 KB de ROM
- 256 KB de RAM
- Conexión a red (fija)
- Soporte completo a la especificación de JVM.
- Interfaz de usuario relativamente limitado.

La configuración CDC está basada en J2SE 1.3.x, incluyendo las siguientes librerías [14]:

Nombre de Paquetes CDC	Descripción
java.io	Clases y paquetes estándar E/S.
java.lang	Clases básicas del lenguaje.
java.lang.ref	Clases de referencia.
java.lang.reflect	Clases e interfaces de reflection.
java.math	Paquete de matemáticas.
java.net	Clases e interfaces de red
java.security	Clases e interfaces de seguridad
java.security.cert	Clases de certificados de seguridad
java.text	Paquete de texto
java.util	Clases de utilidades estándar
java.util.jar	Clases de utilidades para archivos JAR
java.util.zip	Clases y utilidades para archivos ZIP y comprimidos
javax.microedition.io	Clases e interfaces para conexión genérica CDC

Tabla 1: Librerías de la configuración CDC

Connected Limited Device Configuration (CLDC)

Esta configuración está orientada a dispositivos con altas limitaciones computaciones y de memoria, como teléfonos móviles, PDAs, organizadores personales o buscapersonas.

La configuración CLDC está usada por dispositivos con las siguientes características:

- 160 KB a 512 KB de memoria disponible para Java.
- Procesador de 16 a 32 bits, velocidad 8-32 MHz.
- Limitaciones de consumo (baterías).
- Conectividad a red (inalámbrica).
- Restricciones importantes en el interfaz de usuario.

CLDC aporta las siguientes funcionalidades a los dispositivos:

- Un subconjunto del lenguaje Java y todas las restricciones de su Máquina Virtual (KVM).
- Un subconjunto de las bibliotecas Java del núcleo.
- Soporte para E/S básica.
- Soporte para acceso a redes.
- Seguridad

La configuración CLDC incorpora las siguientes librerías [14]:

Nombre de Paquetes CLDC	Descripción
java.io	Clases y paquetes estándar E/S. Subconjunto de J2SE
java.lang	Clases e interfaces de la Máquina Virtual. Subconjunto de J2SE
java.util	Clases, interfaces y utilidades estándar. Subconjunto de J2SE
javax.microedition.io	Clases e interfaces de conexión genérica CLDC

Tabla 2: Librerías de la configuración CLDC

2.4.1.3 Perfiles

Un perfil es un conjunto de APIs orientado a un ámbito determinado. Es decir, un perfil es un conjunto de clases Java que complementan una configuración para un conjunto específico de dispositivos que pertenecen a la misma familia.

Los perfiles definen el ciclo de vida de la aplicación, la interfaz de usuario, etc.

Las librerías de la interfaz gráfica son un componente muy importante en la definición de un perfil. Podemos encontrar grandes diferencias entre interfaces, desde un menú textual en dispositivos móviles a un menú táctil en una PDA.

Un perfil proporciona portabilidad de aplicaciones Java ME entre diferentes dispositivos del mismo segmento vertical o misma familia.

Veamos a continuación los perfiles que existen para cada configuración

Perfiles CDC:

Para la configuración CDC podemos encontrar los siguientes perfiles:

- **Foundation Profile:** Perfil básico para dispositivos sin interfaz gráfico.
- **Personal Basis Configuration:** Perfil gráfico para dispositivos con interfaz gráfico básico.
- **Personal Profile:** Perfil gráfico basado en AWT.

Perfiles CLDC:

Dentro de la configuración CLCD podemos encontrar los siguientes perfiles:

- **Mobile Information Device Profile:** perfil para dispositivos inalámbricos: móviles, PDAs…
- **Information Module Profile:** perfil para dispositivos con interfaz gráfica limitada, como parquímetros o alarmas.

2.4.2 Java Personal Profile

Java Personal Profile es la versión reducida de Java basado en el jdk 1.1.8 para poder ser interpretado por un dispositivo limitado como puede ser una PDA [12] [13].

Aparece en 1997 incorporado en el Personal Profile de la configuración java CDC como un nuevo perfil para permitir desarrollar aplicaciones en dispositivos conectados con interfaces de usuario como set-top boxes.

Personal Java tiene una librería para poder crear interfaces gráficas de usuario (GUI) basadas en java AWT.

2.5 KXML

KXML [15] es un parseador de libre distribución de documentos XML especialmente diseñado para entornos limitados como Applets, Personal Java, o MIDP. Su primera versión nace en Julio de 2000. La versión más actual, lanzada en Diciembre de 2005 es kXML 2 *release* 2.2.2

KXML está basado en los pull parsers con lo que guardan la información de todo el documento parseado.

KXML presenta las siguientes características:

- Soporta el espacio de nombre de XML.
- Tiene un modo para parsear los documentos HTML u otros formatos SGML.
- Bajo requerimiento de memoria.
- Soporta la escritura de documentos XML.

Trabajando con XML:

Existen dos formas de trabajar con XML:

La primera forma es con DOM manipulando el árbol que representa el documento XML. Este árbol se parsea y se guarda en memoria dentro de un nodo. Esta es la forma más sencilla ya que permite buscar todos los elementos y atributos dentro del nodo creado. La desventaja que presenta DOM es que al necesitar guardar toda la estructura del documento, tiene altos requisitos de memoria, sobre todo si el documento XML es muy extenso.

La segunda forma de parsear un documento XML es utilizando SAX. Con este método se capturan los eventos de parseado. Cuando se encuentra un nuevo elemento XML se llama al parser. Por ejemplo, si el parser encuentra una etiqueta como <html>, el escuchador de eventos recibirá el evento y notificará al parser. Al contrario que DOM, no tiene altos requisitos de memoria pero su uso es mucho más complejo.

KXML propone un diseño basado en DOM, no soportando SAX. Sin embargo, al estar orientado a entornos limitados, KXML está diseñado para no necesitar altos requerimientos de memoria y además, no tiene en cuenta los siguientes fallos o excepciones dentro de los archivos XML:

- No detecta los atributos duplicados.
- No descubre la cadena “]]>”
- No entiende la cadena “<?” seguida de un espacio.

También cabe destacar que al igual que kXML, **SAX** [23] es también un parseador óptimo para dispositivos limitados, ya que no guarda en sus elementos toda la información del documento XML, si no que va leyendo poco a poco el usuario según le sea indicado por el programa que lo utiliza. De este modo, los requisitos de memoria que necesita SAX son mínimos.

Capítulo

3

3 Descripción general del sistema

El objetivo principal del monitor de acciones para dispositivos limitados, según vimos en el capítulo 1.3, es desarrollar una aplicación que sea capaz de alertarnos de todas las situaciones que puedan generar ataques peligrosos o que puedan ser dañinos para nuestro dispositivo partiendo de dos fuentes de ataques principales: aplicaciones software instaladas en el terminal y el tráfico de red.

Para lograr dicho objetivo, se desarrollará un sistema basado en lo estudiado en el capítulo dedicado a los IDS:

El monitor de acciones que aquí se presenta es un híbrido entre un IDS basado en red, desde que pretende monitorizar los eventos que se puedan producir desde el interfaz de red y un IDS basado en aplicación, ya que también pretende monitorizar los eventos que produzcan ciertas aplicaciones software Java ME ejecutándose en el dispositivo. Si profundizamos más, también podemos decir que el monitor de acciones es un IDS basado en objetivos ya que le podremos indicar qué aplicaciones son las que queremos que monitorice y cuáles son los eventos que consideraremos como ataque, tanto si presenta la funcionalidad de IDS basado en red o en aplicación.

Para poder desarrollar el monitor de acciones se ha pensado en una arquitectura basada en bloques en la que cada bloque presenta una funcionalidad determinada. Cada bloque a su vez, estará relacionado con otros bloques para de este modo, interaccionar con él y aplicar la funcionalidad del monitor.

3.1 Funcionalidad

En este apartado se presenta la funcionalidad que alcanza y no alcanza el monitor de acciones que se describe en este capítulo.

Veamos cuáles son las funciones que el monitor de acciones sí realiza:

- Permite establecer sesiones de monitorización. Por cada sesión nueva, se genera un fichero de log nuevo.
- Permite reestablecer la última sesión establecida para el monitor de acciones, retomando la ejecución donde se dejó la última vez.
- Permite detectar todos aquellos ataques de red que estén configurados en un registro de ataques potenciales.
- Permite detectar todas aquellas situaciones anómalas que se suceden en aplicaciones Java ME instaladas en el dispositivo.
- Muestra alertas al usuario cuando se ha producido un ataque.
- Registra los ataques detectados en un fichero de log propio para la sesión que se está ejecutando.
- Permite añadir ataques potenciales de red por medio de dos vías: mediante la interfaz gráfica de usuario y modificando el fichero XML en el que se registran, siempre y cuando se aplique el formato del fichero.
- Permite añadir reglas de red o de aplicación, que dirán si un evento es ataque o no, por medio de dos vías: mediante la interfaz gráfica de usuario y modificando el fichero XML en el que se registran, siempre y cuando se aplique el formato del fichero.
- Permite su ejecución en modo *background,* de modo que se podrá seguir trabajando con el dispositivo sin preocuparse de la ejecución del monitor.
- Permite ver los resultados de las últimas sesiones de monitorización.

Estas son las acciones que el monitor de acciones no realiza:

- No evita ataques.
- No toma ninguna acción ante un ataque, sólo alerta. Es decir, no interactúa con ninguna otra aplicación y tampoco toma ninguna acción correctiva o preventiva que pueda evitar que el ataque se produzca de nuevo.
- No repara los daños en el sistema producidos por un ataque.
- No puede compensar mecanismos débiles de identificación y autenticación. Es decir, no se detectan usuarios que suplanten la identidad de otro.
- No puede detectar ataques que no estén configurados en los patrones.
- No puede detectar los eventos maliciosos que se ejecuten en aplicaciones que no hayan sido indicadas al monitor de acciones.

3.2 Arquitectura

Veamos el esquema general de la arquitectura que presentamos para el monitor de acciones:

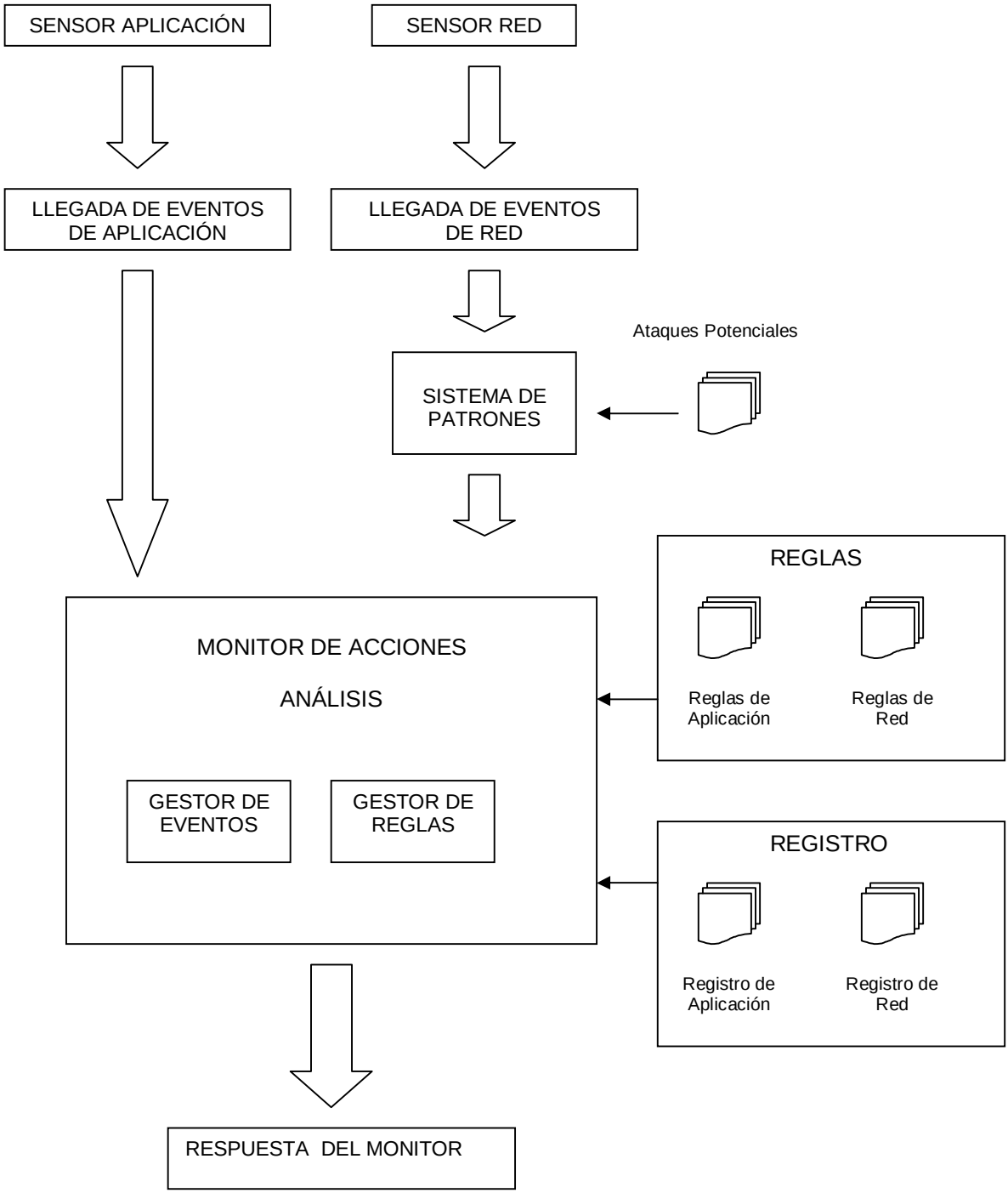


Figura 7: Arquitectura del monitor de acciones

La arquitectura que se presenta en la figura 7 está basada en los dos estándares principales de arquitecturas que podemos encontrar para los IDS:

Por un lado, encontramos todos los elementos principales de una arquitectura CIDF: sensor de eventos, analizador de eventos, registro de eventos y respuesta del IDS. Incluso vemos cómo cada módulo a su vez se relaciona con más de un módulo vecino a la vez, de tal modo que cada componente depende de sus propias acciones y de las salidas de los otros módulos.

Por otro lado, este planteamiento de monitor de acciones se basa en el estándar IDWG en el sentido que utiliza el lenguaje XML para la transferencia de datos entre los módulos registro de eventos, motor de reglas y ataques potenciales. Se ha pensado así debido a las ventajas que los documentos XML pueden proporcionar a una aplicación que esté orientada a dispositivos limitados:

- XML permite la definición de esquemas y DTD y esto nos ofrece la posibilidad de definir la estructura del documento XML según se necesite. Para el monitor de acciones se ha empleado la definición de estructura según esquemas ya que éstos ofrecen mayor abanico de opciones y son más versátiles.
  Al definir la estructura del documento XML, se conseguirá definir un formato de documento de transferencia y registro de datos óptimo orientado a la aplicación, respetando así los requisitos de memoria y espacio en disco que presentan los dispositivos limitados.

- Para parsear los documentos XML dentro de nuestra aplicación Java Personal Profile podremos utilizar el estándar KXML, que como hemos visto en el capítulo 2.4 es uno de los parser más óptimos para dispositivos móviles. De este modo, se alivia la aplicación de carga computacional y de requisitos de memoria.

Analizando los tres modelos de arquitectura que tenemos entre manos (CIDF IDWG y el monitor de acciones) vemos que en todos ellos encontramos tres componentes en común:

- **Sensor de eventos o fuente de datos:** compone el origen de los datos o el origen de los eventos. Gracias a este sensor, podremos analizar todos los eventos que llegan a nuestro dispositivo para posteriormente detectar si dicha entrada de datos es un ataque o no. En el monitor de acciones dividiremos este bloque en dos sub-bloques: sensor de aplicación, que nos va a servir para detectar los eventos que generen las aplicaciones Java ME y sensor de red, para detectar los eventos que se "escondan" dentro del tráfico de red.

- **Analizador:** bloque que traduce la información que llega con los eventos desde el sensor para posteriormente decidir si este evento se considera como ataque o no.

- **Respuesta:** módulo al que llega la salida del analizador. Si el analizador ha detectado un ataque, notificará a este bloque para que elabore y ejecute una respuesta, en función del ataque, de modo que notifique al usuario de la llegada del ataque.

Estos tres módulos formarán el corazón principal de cualquier IDS y, en este caso, el núcleo del monitor de acciones para dispositivos limitados.

Veamos con más detalle la especificación de cada componente que compone la arquitectura del monitor de acciones que aquí se presenta.

3.2.1 Sensor de eventos

El sensor eventos es el primer actor que participa en el sistema. Es aquel módulo que es capaz de detectar toda la actividad que generan las fuentes de datos.

Un **sensor**, según lo define el diccionario de la Real Academia Española de la Lengua es aquel dispositivo que detecta una determinada acción externa y la transmite adecuadamente. Para el monitor de acciones el sensor será el dispositivo (en este caso sensor software) que es capaz de captar toda la información que se produce en nuestro equipo a través de cualquiera de las fuentes de datos que poseemos y la transmite al sistema de patrón de eventos (si la fuente de datos es el tráfico de red) o el motor de análisis (si la fuente es una aplicación software).

El sensor detectará un **evento** cuando la información recibida casa con un conjunto de patrones previamente definidos.

El monitor de acciones tendrá dos tipos de sensores según la fuente de datos ante la que nos encontremos: **sensor de eventos de aplicación**, si es que la fuente de datos es cualquiera de las aplicaciones software Java ME configurada para el monitor o **sensor de eventos de red**, si la fuente de datos es el tráfico de red.

3.2.1.1 Sensor de eventos de aplicación

El **sensor de eventos de aplicación** es el dispositivo o sensor software que detecta la actividad generada por una aplicación software.

El sensor será capaz de leer todos los eventos que se generan dentro de la aplicación.

Las aplicaciones Java ME que se pretenden monitorizar deben generar unos ficheros de log en modo texto que almacenen toda la información que define el evento: hora y fecha del evento, identificador o nombre de la aplicación que genera el evento, usuario que ha generado el evento, identificador único del evento y una descripción opcional del mismo. Dicha información se presenta según el formato que se presenta a continuación:

```
[ fecha día] Event: identificador_del_evento IDUser: identificador_del_usuario Description: descripción_del_evento
```

Cada evento creará una nueva línea en el fichero de log. Veamos un ejemplo de un evento generado por una aplicación:

```
[2008-05-10 18:10:00] Event: Forbidden IDUser: "E2A0363729F0846DA1124D1A2CD3F52B" Description: "evento prohibido"
```

El ejemplo anterior muestra un evento con identificador “Forbidden” que el usuario con identificador “E2A0363729F0846DA1124D1A2CD3F52B” produjo el día 10 de Mayo del 2008 a las seis y diez de la tarde. El evento está descrito con la frase “evento prohibido”.

Cada vez que la aplicación registre un evento en este fichero el sensor de aplicación será capaz de detectarlo. Este evento constituirá la información que el

sensor debe transmitir al motor de análisis del monitor, donde posteriormente se analizará para averiguar si dicho evento constituye o no un ataque.

3.2.1.2 Sensor de eventos de red

El **sensor de eventos de red** es el dispositivo o pequeño sensor software que es capaz de detectar toda la actividad generada por el interfaz de red.

El sensor de eventos de red estará leyendo continuamente todo el tráfico de red que llega al dispositivo.

Cada trama que se recibe o se envía constituirá un evento. Sabemos que en un dispositivo móvil que esté conectado a una red cualquiera se puede detectar mucha cantidad de tráfico de red. Si, como en el caso del sensor de aplicación, se llevara cada evento al motor de análisis, se podría producir una sobrecarga en el monitor de acciones y esto no es deseable. También sabemos que al interfaz de red puede llegar tráfico muy diverso formado por tramas de distintos protocolos. Quizá no nos interese conocer todos los eventos que se producen, si no que nos interesan determinados eventos que presentan unas características determinadas como por ejemplo dirección ip fuente o destino, protocolo, puertos, etc. Por ello, cada trama o evento recibido no se dirigirá directamente al motor de análisis del monitor de acciones si no que previamente se comparará la trama con una serie de patrones que nos dirán si dicho evento se lleva a analizar.

Para capturar todo el tráfico de red que llega a través del interfaz configurado para el monitor de acciones se utilizará la librería Java **Jpcap [16]**. Jpcap es la librería Java usada para capturar y enviar mensajes por la red. En este caso, se utilizará sólo la utilidad de capturar mensajes.

3.2.2 Patrones de eventos: ataques potenciales

Como se ha indicado anteriormente son muchos los eventos que pueden llegar al monitor de acciones. Será interesante aplicar un **filtro** a todos estos eventos para saber cuáles de ellos se pueden considerar como ataque.

Pero, ¿cómo se aplica ese filtro?, ¿cómo se distingue entre un evento cualquiera y un evento que nos interesa monitorizar? La solución consiste en definir un conjunto de patrones que guarden las características de aquellos eventos que sean de interés. Cuando el sensor de eventos detecte un evento cualquiera, mandará dicho evento al patrón y será éste el que compruebe si las características del evento detectado coinciden con cualquiera de los patrones que se tienen almacenados. El evento que casa con uno de los patrones registrados se define como **ataque potencial.**

No tiene sentido definir un patrón de eventos si nuestra fuente de datos es una aplicación Java ME en funcionamiento. Recordemos que un requisito para el monitor de acciones es que las aplicaciones monitorizadas registren sus eventos en sus ficheros de log propios. Esto significa que estas aplicaciones ya están configuradas para loguear los eventos críticos. Teniendo esto en cuenta, si se crea un patrón de eventos de aplicación se estaría añadiendo sobrecarga por varios motivos. Por un lado, se estaría aplicando un filtro a un evento que ya viene filtrado con lo cual, estaríamos realizando más operaciones de las necesarias. Por otro lado, recordemos que se está desarrollando una aplicación para dispositivos limitados, con lo que si se incluye un sensor de eventos de red se añadiría una

clase más que se encargara de este papel y también se incluiría un documento más que guardara la información que representara dichos patrones. Esto no sirve ya que el espacio en disco es limitado y solo se pretende guardar todos los documentos que sean estrictamente necesarios.

Sí se definirá un patrón de eventos para los eventos de red. Los datagramas llegarán sin ningún tipo de filtro inicial, con lo que pueden sobrecargar el sistema. Lo vemos a continuación.

3.2.2.1 Patrones de eventos de red

El subsistema de patrones de eventos de red es un módulo encargado de recibir todos los eventos detectados por el sensor de eventos de red y compararlos con una serie de características que tiene almacenadas para posteriormente concluir en si dicho evento es un ataque potencial o no.

Para poder comparar los eventos que vienen del sensor es necesario tener guardadas unas series de características que indiquen al sistema de patrones si el evento es un ataque potencial o no. Como ya se ha comentado, un sistema óptimo para guardar o transferir información en un dispositivo limitado es el uso de los documentos XML. Por eso, se define un schema para establecer una estructura de documento XML que guarde las características necesarias para detectar ataques potenciales. Dicho schema lo podemos encontrar en el apéndice A.

En el registro XML de ataques potenciales debemos tener almacenadas todas las características que definan un evento de red. En este caso, los eventos de red son datagramas, con lo cual, en el XML deberíamos tener una etiqueta por cada campo que compone un datagrama IP, ICMP, TCP o UDP (ya que son estos protocolos los que se van a monitorizar). Se almacenaran los datos de la cabecera y datos.

A continuación se muestra un ejemplo de un XML que representa los patrones de ataques potenciales:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<PotencialAtacks>
<ip />
<icmp>
      <icmpAtack id="n000001" description="Sale un ping de mi host">
            <tos />
            <length />
            <flags>00</flags>
            <ttl>120</ ttl >
            <srcIP>192.168.1.33</srcIP>
            < />
            <type>8</type>
            <code>0</code>
            <icmpSeq />
        </icmpAtack>
</icmp>
<tcp>
      <tcpAtack id="n000003" description="posible ataque NMAP--> SYN
FLODING">
            <tos />
            <length /> dstIP
            <flags>00</flags>
```

```
            <ttl/>
            <srcIP/>
            <dstIP >192.168.1.33 </dstIP>
            <srcPort />
            <dstPort />
             <seq />
            <ack />
            <tcpFlags>00000010</tcpFlags>
            <window />
      </tcpAtack>
 <udp>
      <udpAtack id="n000005" description="TFTP UDP">
            <tos />
            <length />
            <flags>00</flags>
            <ttl />
            <srcIP />
            <dstIP />
            <srcPort />
            <dstPort>69</dstPort>
            <udpLength />
      </udpAtack>
 </udp>
</PotencialAtacks>
```

En el ejemplo anterior vemos que aparecen tres elementos con los siguientes nombres: icmp, tcp y udp, que corresponden con los protocolos que se desean monitorizar.

Tomemos el primer elemento icmp. Dentro de él se añadirá un nuevo elemento icmpAtack que constituirá un patrón de evento de red ICMP. Se creará un elemento por cada ataque potencial que se desee añadir. Este patrón debe incluir una etiqueta por cada campo que encontremos en un datagrama ICMP. En este caso, se tendrán los campos de cabecera IP más los campos de cabecera ICMP.

Etiquetas de los campos de la cabecera IP:

tos: tipo de servicio del datagrama ip.
length: longitud de datos del datagrama ip
flags: flags del datagrama IP. Se escribirán en formato cadena de texto un flag tras otro en el orden: DF (don't fragment) ,MF (more fragment)
ttl: Time to Life del datagrama.
srcIP: dirección fuente IP. Se escribirá como String en formato aaa.bbb.ccc
dstIP: dirección destino IP. Se escribirá como String en formato aaa.bbb.ccc

Etiquetas de los campos de la cabecera ICMP:

type: tipo de mensaje ICMP
code: código ICMP
icmpSeq: secuencia ICMP

El segundo elemento que encontramos en el ejemplo es tcp. Este elemento englobará a todos los eventos potenciales que sean del protocolo TCP. Al igual que para ICMP, se necesitan todos los campos del datagrama TCP, es decir, tendremos

etiquetas que representen los campos del datagrama IP (las mostradas anteriormente) y etiquetas para los campos del datagrama TCP.

Etiquetas de los campos de la cabecera TCP:

srcPort: puerto origen del datagrama
dstPort: puerto destino del datagrama
seq: campo secuencia del datagrama TCP
ack: valor del campo ack del datagrama TCP
tcpFlags: String que representa a los flags TCP, cada carácter del String representa uno de los flags, siendo 0 si el flan es false o 1 si es true. Los flags irán en el siguiente orden: cwr.ece.urg.ack.psh.rst.syn.fin.

El último elemento representa el protocolo UDP. Dentro de él se incluirán todos los eventos UDP que se consideran ataques potenciales. Las etiquetas que representan los campos de la cabecera UDP son las siguientes:

Etiquetas de los campos de la cabecera UDP:

srcPort: puerto origen del datagrama
dstPort: puerto destino del datagrama
udpLength: longitud de datos del datagrama UDP.

Nota: las etiquetas aparecerán sin rellenar si da igual el valor que tenga el campo
Nota2: la etiqueta flags, siempre irá rellena.

3.2.3 Análisis

Cuando un evento es detectado por el sensor de eventos y es considerado como ataque potencial, llega al motor de análisis. En este módulo se analiza el evento para descubrir si se considera un ataque o no. En el momento en el que se concluya en que el evento es un ataque, se notificará al sistema de logging y alertas para que éste notifique al usuario.

La fase de análisis interactúa directamente con dos módulos del monitor de acciones: con el motor de reglas y el registro de eventos:

- **Motor de reglas:** módulo que se encarga redefinir un conjunto de reglas en formato XML que definirán los ataques que el sistema puede recibir.

- **Registro de eventos:** base de datos en formato XML que almacena todos los eventos que han ido llegando al sistema y no se han considerado como ataque.

Apoyándose en el motor de reglas y registro de eventos el analizador realizará las siguientes tareas:

- **Control de la llegada del evento:** cuando llega un evento desde el sensor de eventos es necesario caracterizar dicho evento, buscando todas sus propiedades creando así un evento con formato válido para el monitor de acciones. El encargado de realizar esta tarea será el submódulo llamando "**Event Manager**" o "**Administrador de Eventos**".

- **Testeo de eventos:** el testeo de eventos lo realizará un submódulo interno al analizador llamando "**Rule Manager**" o "**Administrador de Reglas**". Este testeo consiste en comprobar si el evento casa con un conjunto determinado de reglas que se almacenan dentro del motor de reglas. Se comparan las propiedades del evento que creó "Event Manager" con las propiedades de la regla guardada dentro del "Rule Manager". Si ambas propiedades coinciden significa que el evento detectado es un ataque. En ese caso, se notificará al usuario. La manera de notificación también aparecerá dentro de las propiedades de las reglas almacenadas dentro del administrador de reglas. En el apartado 3.4.1 Motor de Reglas se explicará esto con más detalle.

- **Almacenamiento de eventos:** el almacenamiento de eventos será tarea también de "Event Manager". Cuando un evento capturado por el sensor no cumple las características que se escriben en las reglas, el evento no se considera como ataque. En este caso, el evento es guardado en la base de datos de eventos (registro de eventos) por si posteriormente se pudiera considerar como ataque. Esto es así porque quizá la regla nos dice que el evento se debe producir más de una vez o porque quizá el evento se deba de combinar con otro evento para considerarse un ataque. Si no se registrara la llegada de otros eventos, estas reglas combinadas nunca llegarán a cumplirse.

  El evento no se almacenará sino tiene ninguna regla asociada, es decir, el evento no se considera como ataque.

- **Eliminado de eventos:** si un evento detectado cumple las propiedades que se especifican dentro del motor de reglas y dicho evento ya estaba almacenado en el registro de eventos, dicho evento se elimina del registro.

  Esta operación se realiza por varios motivos: para ahorrar espacio en disco, ya que es un recurso limitado y para poder empezar desde cero por si ese evento es detectado nuevamente por el sensor (se empezarían a contar las nuevas ocurrencias y características del evento).
  El encargado de controlar este eliminado de eventos será también el "Administrador de Eventos"

3.2.4 Registro de eventos

En el Registro de Eventos se lleva a cabo todo el almacenamiento de los eventos que son considerados como ataques potenciales y no casan con alguna de las reglas que se han definido. En general el registro de eventos es la base de datos que posee el monitor de acciones.

El registro está formado por un conjunto de ficheros XML en los que se guarda toda la información de los eventos que forman ataques potenciales y que, tras un análisis, no han sido considerados como ataques.

Sabemos que tenemos dos fuentes de datos: aplicaciones software y tráfico de red. Pues bien, esas dos fuentes generan eventos con información común e información que discrepa, por ello, se desarrollan dos registros distintos para diferencia los dos tipos de eventos: **registro de eventos de aplicación** y **registro de eventos de red.**

Para cada registro XML se han definido unos schemas con los elementos y etiquetas necesarias para cada uno.

3.2.4.1 Registro de eventos de aplicación

En el registro de Eventos de Aplicación es el registro donde se almacenan los eventos de aplicación.

Podemos encontrar el esquema definido para este registro en el apéndice A.

A continuación vemos un ejemplo del registro de eventos de aplicación:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<AplRegister>
<wsfep>
        <user id="E2A0363729F0846DA1124D1A2CD3F52B">
                <event                  id="Forbidden"                  generator="wsfep"
        description="1223907489778">
                        <lastDay>2008-05-10</lastDay>
                        <lastHour>18:00:03</lastHour>
                        <firstDay>2008-05-10</firstDay>
                        <firstHour>18:00:02</firstHour>
                        <frecuency>
                                <times>3</times>
                                <interval>1000</interval>
                        </frecuency>
                        <minInterval>
                                <times>3</times>
                                <interval>500</interval>
                        </minInterval>
                </event>
        </user>
</wsfep>
</AplRegister>
```

A continuación se muestra un análisis del ejemplo de registro de eventos de aplicación que se muestra:

Todo el documento se guarda dentro de un elemento principal llamado **AplRegister** que representa el registro de los elementos. Este elemento contendrá varios elementos que representan cada una de las aplicaciones Java ME que se monitorizan. En el ejemplo vemos que sólo se monitoriza la aplicación WSFEP. A su vez, la aplicación WSFEP contendrá elementos que representan los usuarios que están generando los eventos para dicha aplicación.

Los usuarios de cada aplicación tendrán la siguiente estructura:

- Un atributo que representa su **identificador**.
- Un conjunto de elementos que representan todos los **eventos** que se han ido generando. Para que el registro aporte sentido al sistema, los elemento evento deben guardar toda aquella información que sea necesaria para caracterizar el evento. Así tendremos:
    - Atributos que expresan su identificación (atributo id), la aplicación que lo generó (generator) y una descripción del evento (descripción)
    - Elementos que definen el estado en el que se encuentra el evento. Así tendremos los elementos:

- **lastDay**: nos indica la fecha en la que se produjo el evento por última vez.
- **lastHour**: define la hora en la que el evento se produjo por última vez.
- **firstDay**: nos muestra la fecha en la que el evento se produjo por primera vez.
- **firstHour**: nos enseña la hora en la que el evento se produjo por primera vez.
- **frecuency**: expresa la frecuencia con la que produce el evento, es decir, expresa las veces que se produce un evento dentro de un intervalo de tiempo determinado. Para ello incluye los siguientes dos elementos: times, que es el elemento que nos indica el número de ocurrencias del evento e intervalo, que es el elemento que define el intervalo de tiempo en el que se está produciendo el evento expresado en milisegundos.

  En el ejemplo anterior aparece un evento que se ha detectado 3 veces en un intervalo de 1000 milisegundos.
- **minInterval**: nos muestra el intervalo mínimo en el que el evento se ha producido. Para ello se apoya de dos elementos: times, que expresa el número de veces totales que se ha producido el evento e intervalo, que expresa el mínimo intervalo de tiempo en el que se han producido dos ocurrencias seguidas de un mismo evento.

  En el ejemplo se muestra un evento que se ha producido 3 veces y el menor tiempo entre dos ocurrencias consecutivas del evento ha sido 500 milisegundos.

3.2.4.2 Registro de eventos de red

El registro de eventos guarda en formato XML todos los eventos de red que se deben registrar.

Se define un esquema llamado NetwokRegister.xsd con el formato en el que se almacenan los eventos. Dicho schema se puede ver en el Apéndice A.

A continuación se muestra un ejemplo del registro de red:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<networkRegister>
        <icmp>
                <user id="E2A0363729F0846DA1124D1A2CD3F52B">
                        <event id="n000001" generator="icmp" description="Sale un ping
                        de mi host">
                                <lastDay>2009-05-05</lastDay>
                                <lastHour>17:24:01</lastHour>
                                <firstDay>2009-05-05</firstDay>
                                <firstHour>17:24:00</firstHour>
                                <frecuency>
                                        <times>2</times>
                                         <interval>1421</interval>
                                </frecuency>
                                <minInterval>
                                        <times>2</times>
                                        <interval>1421</interval>
                                </minInterval>
                                <tos>0</tos>
```

```
                        <length>60</length>
                        <flags>00</flags>
                        <ttl>128</ttl>
                        <srcIP>192.168.1.40</srcIP>
                        <dstIP>209.85.229.99</dstIP>
                        <type>8</type>
                        <code>0</code>
                        <icmpSeq>0</icmpSeq>
                    </event>
                </user>
            </icmp>
</networkRegister>
```

En el ejemplo se puede percibir que la estructura del registro de red es muy similar al registro de aplicación, sólo que se añaden unas etiquetas más. Analicemos el ejemplo:

Todo el documento se encierra bajo un elemento principal o raíz llamado **networkRegister** que representa el registro de los eventos. Este es el elemento que equivaldría a **AplRegister** en el registro de eventos de aplicación. Recordemos que este elemento contenía todas las aplicaciones que se monitorizaban, en el ejemplo de registro de aplicación se mostraba a la aplicación Wsfep. Pues bien en este caso el nombre de las aplicaciones se sustituye por el protocolo generador del evento, en el ejemplo que aquí mostramos nuestro **generator** es el protocolo Icmp. Podríamos también tener como generadores al protocolo Tcp o al protocolo Udp.

Siguiendo el paralelismo que existe con el registro de aplicación, a continuación aparecen todos los elementos que definen el estado del evento: `lastTime`, `lastHour`, `firstDay`, `firstHour`, `frecuency` y `minInterval`.

El resto de etiquetas que aparecen son elementos que definen las propiedades de un evento de red, en este caso un evento Icmp. Las etiquetas que aparecen son las mismas etiquetas que existen para el registro de ataques potenciales.

Para todos los protocolos generadores existirán unos elementos en común que corresponden con los campos de la cabecera IP: **tos, lenght, flags, ttl srcIP, dstIP.**

Para cada protocolo se añadirán los elementos correspondientes y característicos de dicho protocolo:

Si el generador del evento es **Icmp**, serán necesarias las siguientes etiquetas: **type, code e icmpSeq**.

Si el generador del evento es **Tcp**, las etiquetas que aparecerán son aquellas que definen a la cabecera Tcp: **srcPort, dstPort, seq, ack, tcpFlags y window**.

Si, por último, quien genera el evento es **Udp**, en el registro aparecerían las etiquetas que simulan los campos de la cabecera Udp: **srcPort, dstPort y udpLenght.**

Todos los elementos anteriormente numerados tendrán el mismo formato que el definido para los ataques potenciales. Solo que a diferencia del registro de

ataques potenciales, en el registro de eventos todos los elementos tendrán un valor. Ese valor corresponderá con el valor que el datagrama presentaba en dicho campo.

3.2.5 Motor de reglas

El motor de reglas del monitor de acciones, junto con el Administrador de Reglas, es el módulo encargado de decidir si los eventos detectados son ataques o no.

Un evento se considera **ataque** si su estado supera unos umbrales determinados definidos dentro de un conjunto de reglas previamente establecido.

Podremos tener distintos tipos de **reglas** en función de las fuentes de eventos:

- **Reglas de Aplicación:** son aquellas reglas que están definidas para los eventos que generan las aplicaciones. Cada regla especifica unos umbrales que un evento determinado no debería superar.

- **Reglas de Red:** forman las reglas de red toda aquella regla que se define para los eventos de red. Cada regla define un conjunto de umbrales que un evento no debe superar.

- **Reglas combinadas:** son aquellas reglas que en su interior combinan más de un evento. Estas reglas, a su vez, se definen en:

    - *Regla combinada de eventos de aplicación:* son reglas de aplicación que incluyen más de un evento de aplicación. Para que se cumpla la regla se deben cumplir todos los eventos que almacena la regla.
      Un ejemplo de esta regla sería que la aplicación software WSFEP genere el evento “x, n veces en un intervalo de tiempo t.
    - *Regla combinada de eventos de red:* es una regla que implica el cumplimiento de varios eventos de red.
      Por ejemplo, se envía una trama del protocolo tcp al puerto x, con dirección destino d.
    - *Regla combinada de eventos de aplicación y red:* este tipo de regla implica que se cumplan uno o varios eventos de red y uno o varios eventos de aplicación.
      Si se combinan los dos ejemplos anteriores generaríamos un ejemplo de este tipo de reglas: se ha producido el evento n de la aplicación WSFEP n veces en un intervalo de tiempo t y posteriormente se envía una trama tcp al puerto x de la dirección ip destino d.

Un evento puede tener más de una regla asociada al igual que existía para el IDS de libre distribución, Snort. Para saber qué regla se debe aplicar en cada caso se establece un algoritmo similar al que aplica Snort.

Cada regla tendrá asociado un número entero, empezando por el cero, que indica la prioridad. La regla más prioritaria es aquella que tiene el menor valor de prioridad. Cuando un evento llega al motor de reglas, se buscan todas las reglas que tiene asociadas y se ordenan de menor a mayor. Una vez se tienen las reglas ordenadas, se va buscando una por una aquella que case con el evento. Si el

evento casa con alguna, se deja de buscar y se aplica la regla. Si el evento no ha casado con ninguna regla no se considera ataque y se descarta.

Para reglas combinadas es el mismo caso, solo que se busca que todos los eventos incluidos en la regla se hayan cumplido con las características especificadas.

3.2.5.1 Formato de las reglas

El monitor de acciones trae por defecto un conjunto de reglas básicas ya definidas. El usuario de la aplicación también podrá definir reglas nuevas, por ello es importante conocer el formato que presentan las reglas y los campos que las definen.

Al igual que para el registro de ataques potenciales y registro de eventos, las reglas se almacenan en un único documento XML. A diferencia que para el registro de eventos en el que se diferenciaba registro de red y aplicación, para el almacén de reglas se crea un único documento XML ya que, como se ha explicado en la clasificación anterior, en una misma regla se pueden combinar eventos de red y aplicación.

El esquema definido para reglas se encuentra en el Apéndice A.

Veamos un ejemplo de documento de registro de reglas y su análisis.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<Rules xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="Rules.xsd">
     <rule action="log/msg" msg="Event Forbiden and session TCP OPEN" priority="0">
          <event id="Forbidden" type="0" generator="Wsfep">
               <frecuency>
                    <times>2</times>
                    <interval>2000</interval>
               </frecuency>
          </event>
          <event id="n000004" type="1" generator="tcp">
               <minInterval>
                    <times>1</times>
                    <interval>0</interval>
               </minInterval>
               <tcpEvent>
                    <tos />
                    <length />
                    <flags>00</flags>
                    <ttl />
                    <srcIP />
                    <dstIP />
                    <srcPort />
                    <dstPort>21</dstPort>
                    <seq />
                    <ack />
                    <tcpFlags>00000000</tcpFlags>
                    <window />
               </tcpEvent>
          </event>
     </rule>
</Rules>
```

El elemento raíz del documento de reglas es <Rules>, éste incluye en su interior uno o más elementos <rule> que equivalen a una regla.

A continuación se muestra la estructura de una regla en el documento XML:

El elemento rule está formado por los siguientes **atributos**:

- **action:** define la acción a realizar en caso de que la regla se cumpla. Las acciones a realizar son las siguientes:
    - log: escribe el evento en el fichero de log.
    - msg: notifica al usuario mostrando un mensaje por pantalla.
    - Log/msg: realiza las dos operaciones anteriores.
- **msg:** muestra en mensaje que se debe enseñar al usuario en caso de que la regla se cumpla.
- **priority:** indica la prioridad de la regla.

El elemento Rule contendrá tantos elementos <event> como eventos se combinen en la regla. Si la regla es sencilla, aparecerá un solo elemento event. En el ejemplo anterior encontramos dos elementos event, esto significa que es una regla combinada. La estructura del elemento event es la siguiente:

- Un elemento event tiene los mismos atributos que tiene su equivalente elemento event en el registro de eventos: **id, type y description.** Esto permitirá fácilmente un evento almacenado en el registro con un evento de una regla.
- El elemento event incluye varios elementos en su interior que especifican los umbrales que no se deben pasar. Dichos umbrales pueden ser de distintos tipos: de **frecuencia** (como se observa en el primer evento del ejemplo) o de **mínimo intervalo** (como aparece en el segundo evento).
  Un umbral de frecuencia define la cantidad de ocurrencias que no debe superar un evento en un intervalo mínimo de tiempo. Por otro lado, un umbral de mínimo intervalo nos indica el mínimo intervalo de tiempo entre eventos. Si este intervalo es menor, significa que el umbral se supera.
  Estos elementos frecuency y minInterval son los mismos que aparecen en el registro de eventos, con lo cual su estructura interior será la misma. Sin embargo, sólo aparecerá uno de ellos: la regla será o de frecuencia o de mínimo intervalo.
- Si el evento que se incluye en la regla es un evento de red, dentro del elemento <event> se incluirá uno de estos cuatro elementos, dependiendo del protocolo origen del evento de red:
    - **<ipInfo>:** incluye todos los elementos que definen la cabecera del protocolo IP. Estos eventos son los mismos que encontramos en el registro de red para el protocolo IP.
    - **<icmpInfo>:** incluye todos los elementos que definen un datagrama icmp. Estos elementos son los mismos que aparecen en el registro de red para el protocolo ICMP.
    - **<tcpInfo>:** incluye todos los elementos que definen un evento tcp. Estos elementos son los elementos que aparecen en el registro de red para un evento TCP.

- **<udpInfo>:** define los elementos que componen un datagrama udp. Los mismos eventos los encontramos en el registro de red para los eventos UDP.

Nota: al igual que para los ataques potenciales, si uno de los elementos mostrados anteriormente aparece vacío, significa que la característica que define ese elemento no se tiene en cuenta. Es decir, da igual el valor que tenga el evento detectado para ese campo del datagrama.

3.2.6 Sistema de loggings y alertas

El sistema de Loggings y Alertas es el sistema encargado de notificar al usuario en el caso de que se haya producido un ataque. Constituye la salida o respuesta del monitor de acciones.

Para notificar al usuario tenemos tres posibles maneras:

- **Alertar:** abre una ventana de diálogo como la que se muestra a continuación indicando al usuario que se ha producido un ataque:

Figura 8: Ejemplo de alerta de ataque

- **Loguear:** loguea el ataque en un fichero de log. En el fichero de log se guardará: fecha y hora en la que se produjo el ataque, fuente que lo produjo (nombre de la aplicación o protocolo de red), identificador del evento, identificador del usuario que lo produjo, descripción del evento y mensaje de alerta que aparece dentro de la regla XML.
  Veamos un ejemplo de un evento logueado en el log del monitor de acciones:

  ```
  [ jue abr 23 22:06:40 2009 ]  Fuente: icmp Evento: n000001 Usuario: E2A0363729F0846DA1124D1A2CD3F52B Descripción: Sale un ping de mi host msg: Ping Flooding attack from host
  ```

  Cada vez que se inicie una nueva sesión en el monitor de acciones se creará un nuevo fichero con la fecha de inicio de sesión.

- **Alertar y Loguear:** esta es la acción más común ya que engloba las dos anteriores. Por un lado muestra la alerta visual y por otro lado, loguea el evento en fichero de log del monitor de acciones

El sistema de *logging* y alertas debe interactuar con el motor de reglas para poder identificar la acción que debe llevar a cabo ya que es en el fichero XML de reglas donde se especifica la respuesta del monitor a aplicar en caso de que un evento case con la regla y el mensaje de log o de alerta a mostrar.

3.3 Restricciones y Suposiciones

Para que se pueda proporcionar la funcionalidad especificada para el monitor de acciones es necesario tener en cuenta lo siguiente:

Las aplicaciones que se deseen monitorizar deben ser capaces de almacenar en un fichero de log los ataques que ellas mismas generen. Además de almacenarse, deben hacerlo en el formato que se ha especificado en el sensor de eventos de aplicación.

El monitor de acciones asume que el dispositivo limitado en el que corre le ofrece los permisos necesarios para escuchar los eventos de red. Es decir, se debe poder escuchar el tráfico de red sin ningún tipo de restricción.

Se debe contar con espacio en disco libre para poder escribir la información necesaria. En apéndice B se muestran con más detalles los requisitos de espacio en disco necesarios.

Capítulo

4

4 Implementación del Monitor de Acciones

Tras haber especificado las funciones y requisitos del monitor de acciones se realiza la implementación del mismo apoyándose en el modelo de la arquitectura mostrado en el capítulo 3.

El monitor de acciones se desarrollará por medio de la tecnología Java. Dentro de todas las posibilidades que Java ofrece se utilizará la tecnología Personal Profile, la cual se incluye en la configuración CDC que es una de las dos configuraciones de Java 2 Micro Edition (Java ME). Java Personal Profile, según se estudia en el capítulo 2.4, proporcionará todas las librerías básicas que van a permitir desarrollar el monitor de acciones y además, proporcionará un paquete (java.awt) para poder desarrollar la interfaz gráfica con la que interactuará el usuario.

Aún así, Java Personal Profile se queda corto y es necesario utilizar las librerías KXML para la lectura y escritura de los documentos XML que forman la base de datos del monitor y la librería JPCAP [16] para poder leer y analizar todo el tráfico de red que llega al dispositivo.

4.1 Visión general del Monitor de Acciones

Tal y como se presenta en el capítulo 3, la arquitectura del monitor de acciones está formada por distintos bloques o módulos en los que cada bloque o módulo tiene una función definida para el mismo. Pues bien, para desarrollar el monitor de acciones se utiliza la misma filosofía creando así varios paquetes Java que emulan los módulos de los que se compone el monitor de acciones. Cada paquete tendrá la misma función que la que desempeña el módulo al que representan.

Además de los paquetes Java Personal Profile que implementan los módulos del monitor de acciones, se han creado otros paquetes que no tienen que ver directamente con el monitor pero que sirven para apoyar a los paquetes "módulo" anteriores.

Todos los paquetes desarrollados se incluyen dentro de un paquete global llamado **`es.uc3m.it.monitor.`** Veamos cuáles son los paquetes:

- **Paquete funcionalidades**: este paquete contiene todas las clases que representan a Java Beans. En realidad no corresponde con ningún módulo pero es necesario para poder establecer las unidades básicas o los actores de la aplicación (como por ejemplo, los eventos)

- **Paquete gestiones**: paquete que representa el módulo principal de análisis del monitor de acciones incluyendo al administrador de eventos y al administrador de reglas.

- **Paquete KXML**: paquete que almacena todas las clases que nos permitirán leer y escribir en los documentos XML en los que se apoya la aplicación. Es decir, representa las clases que accederán a la base de datos. Así de este modo, este paquete representará simultáneamente al registro de eventos, al registro de reglas y al registro de ataques potenciales.

- **Paquete sniffer**: es el paquete que forma o corresponde con el sensor de eventos de red.

- **Paquete wsfep**: es el paquete que representa al sensor de eventos para la aplicación Wsfep, que es la aplicación de prueba utilizada para este proyecto.

- **Paquete utilidades**: este paquete no corresponde con ningún módulo del monitor de acciones. Es un paquete definido para incluir todas las utilidades y clases de apoyo que no tienen que ver directamente con la aplicación.

- **Paquete GUI**: Este paquete está creado para incluir todas las clases que tienen que ver con la interfaz gráfica de la aplicación. Permitirá al usuario configurar nuevas reglas y ataques potenciales y formará el módulo de sistema de loggings y alertas ya que muestra el resultado del análisis de los eventos. Este paquete se verá con más profundidad en el capítulo 5.

A continuación se describe con detalle la funcionalidad de cada uno de los paquetes y se muestra un diagrama UML que indica cómo se relacionan unas clases con otras y qué atributos y métodos define cada clase.

4.1.1 Paquete funcionalidades

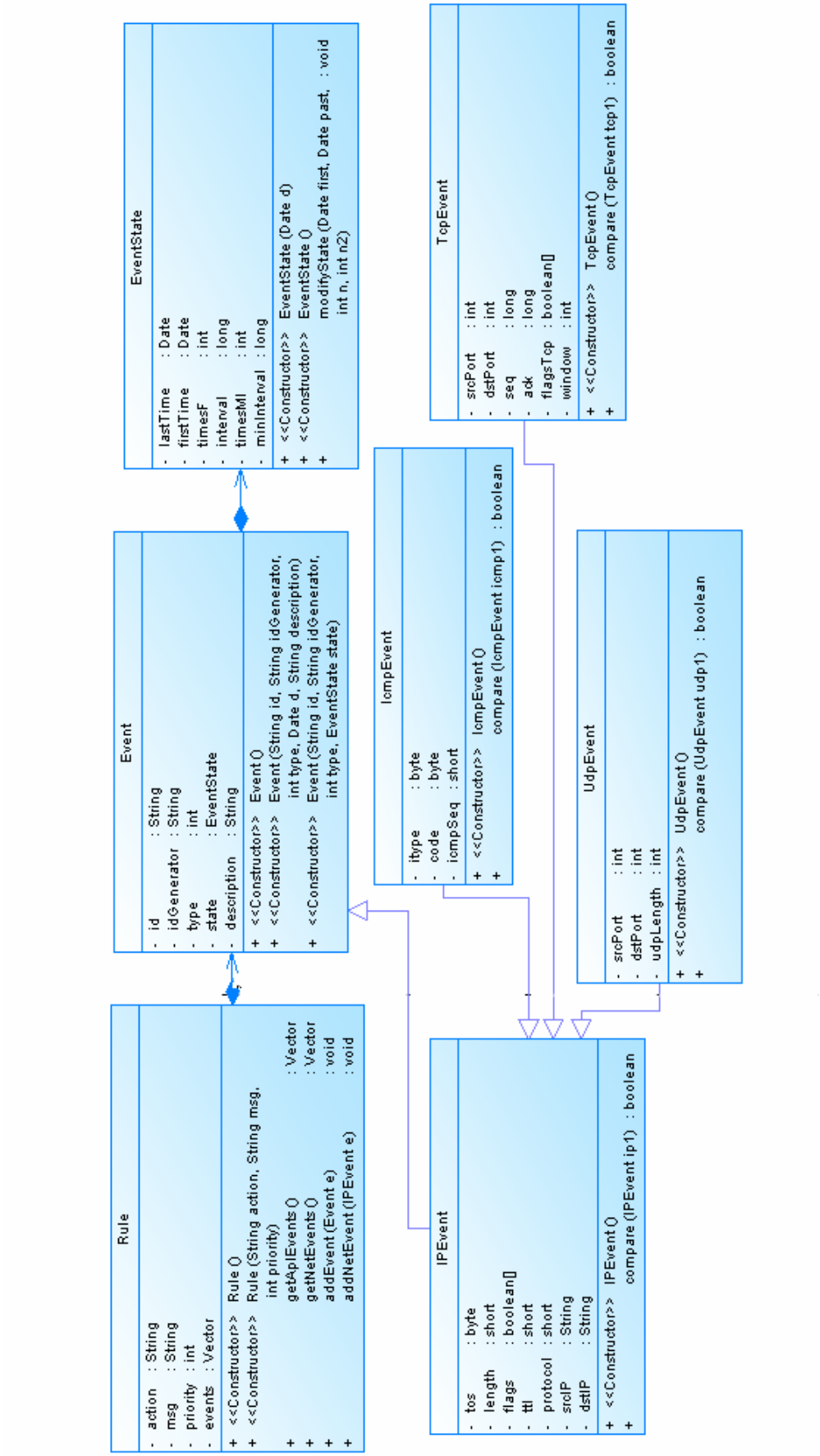


Figura 9: Diagrama UML del paquete funcionalidades

Este paquete está formado por todas las clases Java Bean que representan a los elementos más pequeños del monitor de acciones. Las clases que aquí se incluyen permiten crear objetos que serán los actores que participen en la aplicación. Estos actores son los eventos de aplicación y red, los estados de los eventos y las reglas.

A continuación se muestra el análisis y función de cada clase:

❖ **Clase Event**

Esta clase representa a un evento detectado por cualquiera de los sensores del monitor de acciones. Incluye todas las características comunes que tienen los eventos de aplicación y los eventos de red.

Un evento de aplicación se podría definir perfectamente por esta clase, sin embargo para definir un evento de red se necesitarán otras características relacionadas con el tráfico de red que se incluirán en otras clases distintas que se mostrarán a continuación.

El evento de aplicación, o parte de un evento de red, está definido por los siguientes atributos:

- **Id:** es una cadena de texto que representa el identificador del evento. Cada evento tendrá un id único de tal modo que el id será como el "DNI" del evento.

- **idGenerator:** es una cadena de texto que representa el identificador único de la aplicación software que generó el evento. Un ejemplo de idGenerator puede ser "wsfep". Wsfep, es la aplicación que se monitoriza en el monitor de acciones. Para el caso de eventos de red, los posibles generadores que se utilizarán corresponderán con los nombres de los protocolos monitorizados: "ip", "icmp", "tcp" o "udp".

- **Type:** es un número entero que indica si el evento ha sido generado por una aplicación o por un protocolo de red. Es decir, este atributo distingue entre las dos fuentes posibles de eventos. Type tendrá el valor de 0 si la fuente del evento es una aplicación software y tendrá el valor 1 si la fuente del evento ha sido el interfaz de red.

- **State:** de tipo EventState representa el estado del evento: número de veces que se ha producido, fecha de la última y la primera vez que se produjo e intervalo de tiempo en el que se ha producido.
  Se mostrará más información de este atributo en la definición de la clase EventState.

- **Description:** es una cadena de texto que muestra una descripción explicativa sobre el evento. Si el evento es de aplicación, la descripción del evento vendrá dada por la aplicación. Sin embargo, si el evento es de red la descripción del evento se definirá dentro del documento XML de registro de ataques potenciales.

Los métodos que contiene esta clase son aquellos que permiten crear nuevos objetos de este tipo y que nos permiten dar valor y tener acceso a los atributos.

❖ **Clase EventState**

Es la clase que representa el estado en el que se encuentra un evento en un momento dado. El estado que representa esta clase puede pertenecer tanto a un evento de red como a un evento de aplicación.

Los atributos que se incluyen en esta clase son los atributos que definen la situación actual del evento. A continuación se definen estos atributos:

- **lastTime:** de tipo Date, representa la fecha en la que el evento se produjo por última vez.

- **firstTime:** también de tipo Date, representa la fecha en la que el evento se produjo por primera vez.

- **timesF:** es un número entero que indica las veces que se ha producido un evento en un determinado intervalo de tiempo.

- **Interval:** de tipo long representa el intervalo de tiempo en el que se ha producido el evento. Es decir, es la diferencia entre la primera ocurrencia del evento y la última. Junto con timesF, intervalo define la frecuencia del evento.

- **timesMI:** es un número entero que indica el número de veces que el evento se ha producido dentro de un intervalo mínimo de tiempo que el evento no debe disminuir.

- **minInterval:** de tipo long representa el mínimo intervalo de tiempo que un evento no debe disminuir. Junto con timesMI, minInterval define la frecuencia de mínimo intervalo del evento. Es decir, expresa el mínimo intervalo de tiempo que no se debe disminuir para que el evento no se considere un ataque.

Una vez definidos todos los atributos de esta clase, se especifican métodos que construyen objetos de esta clase y métodos que permiten acceder y modificar el valor de los atributos tal y como se define para cualquier clase Java Bean.

Además de las operaciones anteriores, de define una operación más:

- **modifyState (Date first, Date past int n1, int n2):** es una operación que permite cambiar el estado de un evento. Incrementa el número de veces en la que se produce el evento y recalcula el intervalo de tiempo en el que se produce el evento, tanto si es un intervalo de frecuencia como si es un mínimo intervalo.

  El método recibe los siguientes parámetros:

  - Date first: primera vez en la que el evento se produjo.
  - Date past: última fecha en la que se generó el evento.
  - Int n1: número de veces en la que el evento se produjo dentro de un intervalo de tiempo que expresa frecuencia.
  - Int n2: número de veces que un evento se ha producido dentro de un intervalo mínimo de tiempo.

❖ **Clase `IPEvent`**

Esta clase representa a los eventos de red. Hereda de Event ya que un evento de red es un tipo de Evento y como tal necesita tener los atributos que le definen y un estado.

La clase IPEvent que representa a un datagrama IP, engloba a todos los eventos de red posibles ya que dentro del protocolo IP se encapsulan el resto de protocolos que se monitorizan: Icmp, Tcp y Udp.

Al representar a un paquete IP, la clase IPEvent debe contener los atributos que representen a este protocolo y que sean comunes para los protocolos encapsulados dentro de cualquier paquete IP. Veamos cuáles son estos atributos:

- **Type**: byte que representa el tipo de servicio del datagrama IP.
- **Length:** de tipo short indica la longitud de datos del paquete IP.
- **Flags:** es un array de booleanos en el que cada posición del array se almacena un flag del datagrama IP. Si el flag es 0, el booleano será false, si por el contrario es 1, el booleano será true.
  Los flags se guardan dentro del array en el siguiente orden:
  - En la posición 0 del array se almacena el flag don't frag que indica que el paquete no está fragmentado.
  - En la posición 1 del array se guarda el flag more frag del datagrama IP que indica si ese paquete es o no el último fragmento de una trama enviada.
- **Ttl:** de tipo short indica el tiempo de vida o número de saltos que puede dar una trama IP.
- **Protocol:** de tipo short indica el protocolo que va encapsulado en el campo de datos del datagrama. Se van a tener en cuenta los protocolos:
  - **1-**ICMP
  - **6-**TCP
  - **17-**UDP
- **srcIP:** cadena de caracteres que representa la dirección IP de origen de la trama IP detectada por el sensor de eventos de red.
- **dstIP:** cadena de caracteres que indica la dirección IP destino de la trama IP recibida.

Entre los métodos de la clase IPEvent se incluyen varios constructores y todos los métodos que nos permiten acceder y modificar los atributos de la clase, que son los métodos que debe tener la clase por ser un Java Bean. Además de esos métodos obligatorios, la clase IPEvent incluye el método:

- **boolean compare (IPEvent ip1):** método que compara dos eventos Ip. Comprueba todos los campos uno a uno para saber si los dos eventos son iguales o no. En caso de que los eventos sean iguales, el resultado del método será un booleano con valor true. En caso contrario, devolverá el mismo booleano con valor negativo.
  Este método sólo tiene un parámetro de entrada:
  - IPEvent ip1: Evento IP con el que queremos comparar el evento actual.

❖ **Clase `IcmpEvent`**

La clase IcmpEvent representa un evento producido por el protocolo Icmp. Icmp es un protocolo que va encapsulado dentro del datagrama IP, por ello un evento Icmp, además de presentar sus propias características, debería tener aquellas características que definen al protocolo IP. Teniendo en cuenta lo anterior, IcmpEvent heredará de la clase IPEvent para que así pueda disponer de los atributos que definen a las clases Event e IPEvent y además de los atributos que le definen a él:

- **Itype:** especifica el tipo de mensaje Icmp que es. De tipo byte, itype toma los mismos valores que el campo type del mensaje ICMP:
  - 0 – Echo reply
  - 3 – Destination unreachable
  - 4 – Source quench
  - 5 – Redirect
  - 8 – Echo
  - 9 – Router advertisement
  - 10 – Router solicitation
  - 11 – Time exceded
  - 12 – Parameter problem
  - 13 – Timestamp request
  - 14 – Timestamp reply
  - 17 – Address mask request
  - 18 – Address mask request
- **Code: d**e tipo byte, contiene el código de error para el datagrama del que da parte el mensaje Icmp.
- **imcpSeq:** de tipo short es el número de secuencia del datagrama Icmp.

Al igual que el resto de clases que comparten paquete con IcmpEvent, esta clase tiene todos los métodos necesarios para crear objetos de este tipo y para acceder y modificar los atributos de la clase.

Se añade un método más que se muestra a continuación:

- **boolean compare (IcmpEvent icmp1):** este método, al igual que su método homónimo en la clase IPEvent, se encarga de comparar el evento Icmp con el evento que es pasado por parámetro.
  En este método se comprueba que todos los atributos sean iguales. En caso afirmativo se concluye en que ambos eventos ICMP son iguales y se devuelve un booleano con valor positivo, en caso contrario, se devuelve el mismo booleano solo que con valor negativo.

❖ **Clase `TcpEvent`**

Esta clase representa un evento de red que llega encapsulado en un datagrama Tcp.

Al igual que la clase IcmpEvent, TcpEvent hereda de IPEvent para poder tener todas las cualidades comunes que tiene cualquier evento de red. Para poder distinguirlo entre cualquier evento de red, la clase TcpEvent tiene una serie de atributos que coinciden con los campos más importantes de un paquete TCP. Veamos cuáles son esos atributos:

- **srcPort:** número entero que indica el puerto origen del datagrama Tcp
- **dstPort:** número entero que indica el puerto destino del datagrama Tcp
- **seq:** de tipo long, indica el número de secuencia del paquete TCP. Este campo sirve para comprobar que ningún paquete se ha perdido y que llegan en el orden correcto.
- **Ack:** de tipo long, es el número de acuse de recibo. Si el flag ack está activo, este campo indicará el número de secuencia del siguiente byte que se espera recibir.
- **tcpFlags:** array de booleanos en el que en cada posición se almacena un flag del datagrama Tcp. Si el valor del flag es 1, el booleano almacenado será positivo, si por el contrario, el flag es 0, el booleano tendrá valor negativo. A continuación se muestra el orden en el que se almacenan los flags dentro del array.
    - En la posición 0 del array se almacena el flag cwr o "Congestion Window Reduced". Este flag se activa para indicar que se ha recibido un paquete con el flag ECE activado.
    - En la posición 1 del array se almacena el flag ECE que indica que el receptor puede realizar notificaciones ECN.
    - La posición 2 del array es ocupada por el flag URG que indica que el campo urgente del datagrama TCP es significativo.
    - En la posición 3 encontraremos el flag ack que indica que el paquete es un paquete ack.
    - En la posición 4 del array aparece el flag psh que indica que los datos almacenados en el buffer deben ser transferidos.
    - La posición 5 del array es la que contiene el flag rst que si está activo indica que la conexión se termina sin esperar respuesta.
    - En la posición 6 del array se almacena el flag syn que activa o desactiva la sincronización de los números de secuencia.
    - Por último, en la posición 7 del array se guarda el valor del flag FIN que se activa si no hay más datos que enviar.
- **Window:** número entero que indica el número máximo de bytes que el receptor está esperando recibir.

Entre los métodos definidos para esta clase se encuentran, al igual que para el resto de clases, aquellos que nos permiten crear nuevos objetos de tipo TcpEvent y acceder o modificar sus atributos.

Además podemos encontrar el método **compare** que compara dos eventos Tcp e indica si los eventos son iguales o no siguiendo la filosofía que este método aplica en las clases IPEvent e IcmpEvent.

❖ **Clase `UdpEvent`**

Clase que representa un evento de red producido por el protocolo Udp. Al ser Udp un protocolo que se encapsula dentro del protocolo IP un evento Udp debería tener todas las cualidades de un evento IP más otras cualidades que definan a un evento Udp. Por ello, la clase UdpEvent hereda de IPEvent.

Los atributos que definen a un evento Udp y que se encuentran en la clase UdpEvent son los siguientes:

- **srcPort:** número entero que indica el puerto origen del datagrama Udp.
- **dstPort:** número entero que indica el puerto destino del datagrama Udp.
- **udpLength:** número entero que indica la longitud del campo de datos del datagrama Udp.

La clase UdpEvent, al igual que IPEvent, IcmpEvent y TcpEvent, contiene el método **compare** que compara dos eventos Udp devolviendo como resultado un booleano que indica si los eventos son iguales o no.

Además del método compare, la clase UdpEvent, contiene los constructores que permiten crear nuevos objetos UdpEvent y métodos que permiten conseguir y modificar el valor de los atributos de la clase.

4.1.2 Paquete gestiones

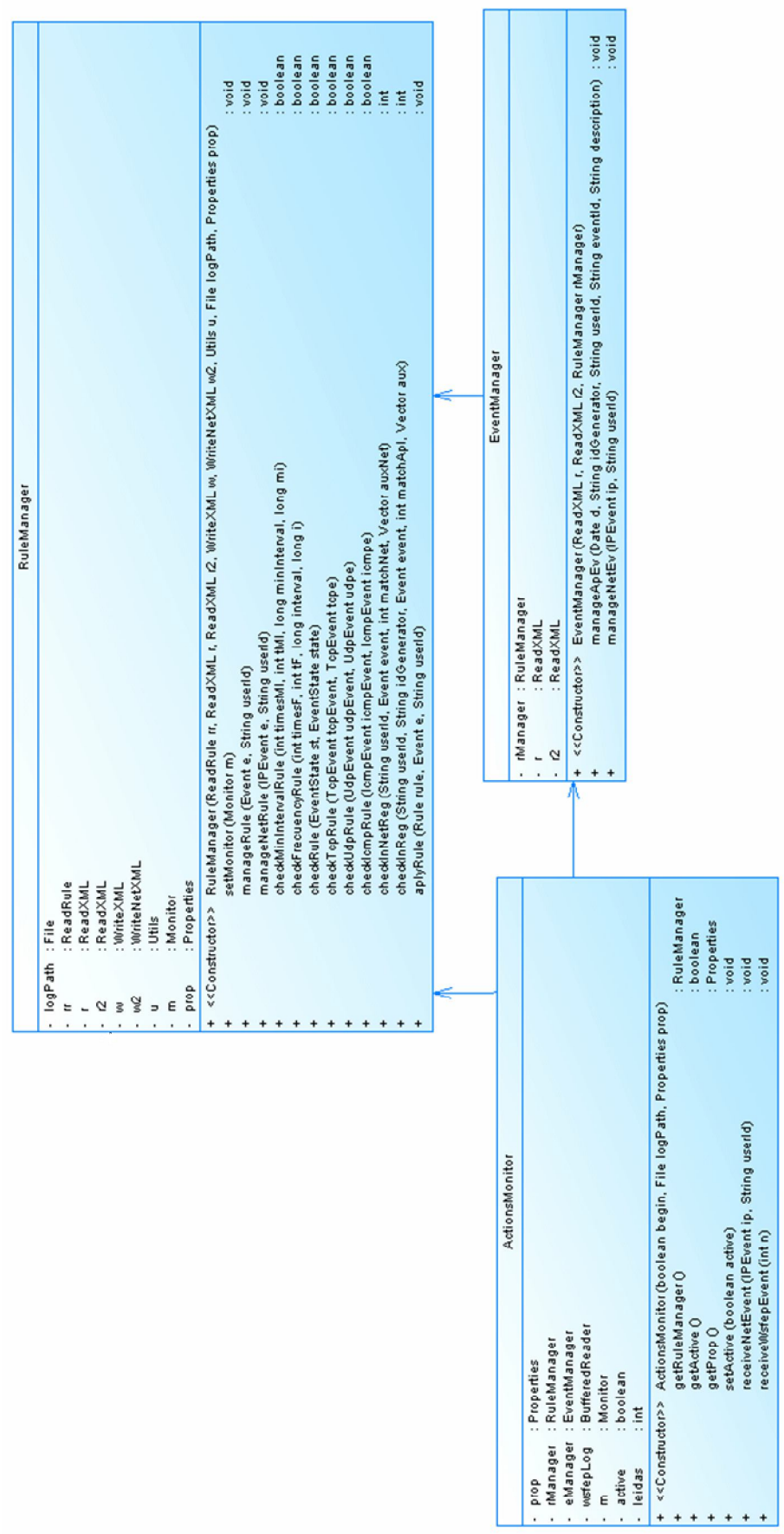


Figura 10: Diagrama UML del paquete gestiones

El paquete gestiones contiene las clases que administran las reglas y los eventos para poder desarrollar el papel del módulo de análisis del monitor de acciones.

Veamos las clases de las que se compone este paquete:

❖ **Clase `ActionsMonitor`**

Es la clase que forma el núcleo o centro de la aplicación. El monitor de acciones es la clase a la que llegan todos los eventos de red o aplicación que son considerados como ataques potenciales.

Una vez recibido el evento, ActionsMonitor se encarga de desglosar el evento y llevarlo al administrador de eventos y administrador de reglas para analizarlo e identificar si el evento es ataque o no.

A continuación se muestran los atributos más destacados de la clase:

- **Properties prop:** Properties para poder acceder al fichero de propiedades en el que se almacenan las rutas de los ficheros más importantes que se utilizan a lo largo de la aplicación, como por ejemplo, el fichero de reglas, el fichero de registro de eventos de aplicación o de red, el fichero de ataques potenciales, el fichero de log de las aplicaciones utilizadas o el fichero de log resultado del monitor de acciones.

- **BufferedReader wsfep:** clase para poder leer el fichero de log de la aplicación que se monitoriza en el monitor de acciones.

- **Monitor m:** representa a la ventana principal de la interna gráfica dónde se ejecuta el monitor de acciones y se muestran los resultados.

- **Boolean active:** booleano que indica si el monitor de encuentra en funcionamiento, es decir, que está activo o si el monitor está parado o desactivado. En este caso no tendrán en cuenta ninguno de los eventos que lleguen.

Entre los métodos que contiene ActionsMonitor encontramos:

- **ActionsMonitor (boolean begin, File logPath, Properties prop):** es el constructor de la clase. Crea un nuevo objeto ActionsMonitor para poder comenzar a monitorizar.
  El método recibe los siguientes parámetros:

  - Boolean begin: indica si se inicia una nueva sesión o si se continúa con una monitorización anterior.
    Iniciar una sesión supone eliminar todos los eventos que estaban almacenados en el registro de eventos y crear un nuevo fichero de log de eventos.
  - File logPath: File que representa al fichero en el que se loguean los ataques detectados.
  - Properties prop: propiedades que tendrá ActionsMonitor.

- **void receiveNetEvent (IPEvent e, String userId):** método al que se invoca cuando se ha detectado un ataque potencial de red. Se encarga de parsear el evento y llevarlo al administrador de eventos de red para que éste le pueda analizar.

El método recibe los siguientes parámetros:

- IPEvent e: evento IP considerado como ataque potencial
- String userId: identificador del usuario que ha generado el evento.

Este método no devuelve ningún resultado.

- **void receiveWsfepEvent ():** este método es invocado cuando se detecta actividad en el fichero de log de la aplicación Wsfep. Esto quiere decir que se ha logueado un nuevo evento para esta aplicación y ActionsMonitor lo debe tratar.
  Para ello, receiveWsfepEvent lee la nueva o las nuevas líneas añadidas en el fichero y crea un nuevo objeto Event por cada evento registrado en el log de Wsfep.
  Tras haber creado el o los eventos Event, los lleva al administrador de eventos de aplicación para que éste los analice.
  Este método no recibe ningún parámetro ni devuelve ningún resultado.

❖ **Clase `EventManager`**

Clase que representa al administrador de eventos de red y al administrador de eventos de aplicación. Recibe los eventos del ActionsMonitor para analizarlos, tratarlos y presentárselos al administrador de reglas.

La clase presenta los siguientes atributos:

- **RuleManager rManager:** administrador de reglas con el que interactúa la clase. A él le transferirá los eventos que analice.

- **ReadXML r:** clase que nos permite leer el documento XML en el que se registran los eventos de aplicación.

- **ReadXML r2:** clase que permite acceder a los eventos que se almacenan en el registro de eventos de red en formato XML.

Los métodos incluidos en esta clase son los siguientes:

- **manageAplEvent (Date d, String idGenerator, String userId, String eventId, String description):** este método recibe todos los datos que detecta el sensor de eventos de aplicación. Convierte los datos recibidos en un objeto de tipo Evento que posteriormente enviará al motor de reglas.
  Los parámetros que recibe este método son los siguientes:

  - Date d: fecha en la que se ha producido el evento.
  - String idGenerator: identificador de la aplicación a la que pertenece el evento generado.
  - String userId: identificador del usuario que generó el evento.
  - String eventId: identificador del evento.
  - String description: descripción del evento

- **manageNetEvent (IPEvent ip, String userId):** recibe los eventos que son detectados por el sensor de eventos de red y lo transmite al motor de reglas. Este método recibe los siguientes parámetros:

  - IPEvent ip: evento de red detectado por el sensor.

- String userId: identificador del usuario que generó el evento.

❖ **Clase `RuleManager`**

Clase que representa al motor de reglas del monitor de acciones. Para realizar esta labor se apoya de los siguientes atributos:

- **File logPath:** representa el fichero de log en el que se registran los eventos que superan los umbrales definidos por las reglas. Es decir, en este fichero se loguearán todos los eventos que se consideren ataques.

- **ReadRule rr:** objeto ReadRule que permite leer la información almacenada en el registro de reglas.

- **ReadXML r:** permite leer la información de los eventos registrados en el registro de eventos de aplicación.

- **ReadXML r2:** permite leer la información de los eventos almacenados en el registro de eventos de red.

- **WriteXML w:** registrará los eventos de aplicación que no superen los umbrales de las reglas en el registro de eventos de aplicación.

- **WriteXML w2:** ayuda a almacenar los eventos de red que no superan los umbrales de las reglas en el registro de eventos de red.

- **Monitor m:** ventana principal del monitor de acciones donde se muestran los ataques producidos.

- **Properties prop:** propiedades del sistema que indican la localización de los ficheros utilizados.

En el motor de reglas destacan los siguientes métodos:

- **RuleManager (ReadRule rr, ReadXML r, ReadXML r2, WriteXML w, WriteNetXML w2, Utils u, File logPath):** construye un nuevo objeto RuleManager.

- **manageRule (Event e, String userId):** método principal de la clase que se encarga de administrar la llegada de un evento al motor de reglas. Comprueba si el evento tiene reglas asociadas a él en el registro de reglas. Si el evento tiene reglas almacenadas pide que se compruebe si el evento y el resto de eventos que se muestran en la regla (si es que es una regla combinada) superan los umbrales de las reglas. Si el evento no tiene reglas, pide que se añada el evento al registro de red.
  Los parámetros que recibe este método son los siguientes:
  - Event e: evento que llega al motor de reglas y al que se le deben buscar reglas asociadas para saber si es ataque o n.
  - String userId: identificador del usuario que generó el evento.

- **manageNetRule (IPEvent e, String userId):** realiza las mismas operaciones que se explican para el método manageRule solo que para eventos de red.
  Los parámetros que recibe el método son los siguientes:
  - IPEvent e: evento de red que llega al motor de reglas.
  - String userId: identificador del usuario que genera el evento.

- **checkRule (EventState st, EventState state):** método que comprueba si un evento supera los umbrales que se especifican en la regla que tiene asociada. Devuelve un booleano con valor positivo si el evento supera los umbrales. Si no los supera, devolverá un booleano falso.
  Este método recibe los parámetros:
  - EventState st: estado del evento que llega al motor de reglas.
  - EventState state: estado almacenado en la regla asociada al evento. Los atributos del estado anterior no deben superar los umbrales definidos en los atributos de este estado.

- **checkTcpRule, checkUdpRule, checkIcmpRule:** se engloban estos tres métodos en el mismo apartado ya que realizan operaciones similares. Comprueban si los atributos de un evento de red coinciden o no con los atributos que se almacenan en la regla asociada a ese evento. Se define un método por cada protocolo de red monitorizado ya que en cada protocolo se definen características diferentes.

- **checkInNetRegister (String userId, Event event, int matchNet, Vector auxNet):** este método es consultado cuando el evento que llega al motor de reglas tiene una regla combinada. Este método trabaja con uno de los eventos de red con los que se combina el evento principal. Se comprueba si ese evento está almacenado en el registro de red. Si está almacenado, obtiene su estado y comprueba si ese estado supera los umbrales definidos en la regla. Si es así, incrementa el valor de un número entero pasado por parámetro y lo devuelve como resultado del método.
  Los parámetros recibidos por el método son los siguientes:
  - String userId: identificador del usuario que generó el evento.
  - Event event: evento que se busca en el registro de red.
  - Int matchNet: número de eventos de red que se combinan con el evento que llega al motor de reglas y que han sobrepasado los umbrales definidos en la regla. Este valor se incrementa en 1 si este evento de red supera los humbrales de la regla.
  - Vector auxNet: vector que guarda el nombre y la descripción del evento de red.

- **checkInRegister (String userId, String idGenerator, Event event, int matchApl, Vector aux):** método que realiza las mismas operaciones que el anterior solo que aplicadas a eventos de aplicación que se combinan con el evento principal.

- **aplyRule (Rule rule, Event e, String userId)**: este método es invocado cuando el evento que llega y todos los eventos que se combinan con él dentro de la regla (si es que los hay) superan todos los umbrales de la regla. En este momento el evento se considera ataque y es necesario aplicar la acción que indica la regla.
  Este método es quien hace las funciones de sistema de loggings y alertas.
  Los parámetros que recibe son los siguientes:
  - Rule rule: regla que se debe aplicar.
  - Event e: evento que se ha producido y se ha considerado como ataque.
  - String userId: identificador del usuario que generó el ataque.

4.1.3 Paquete KXML

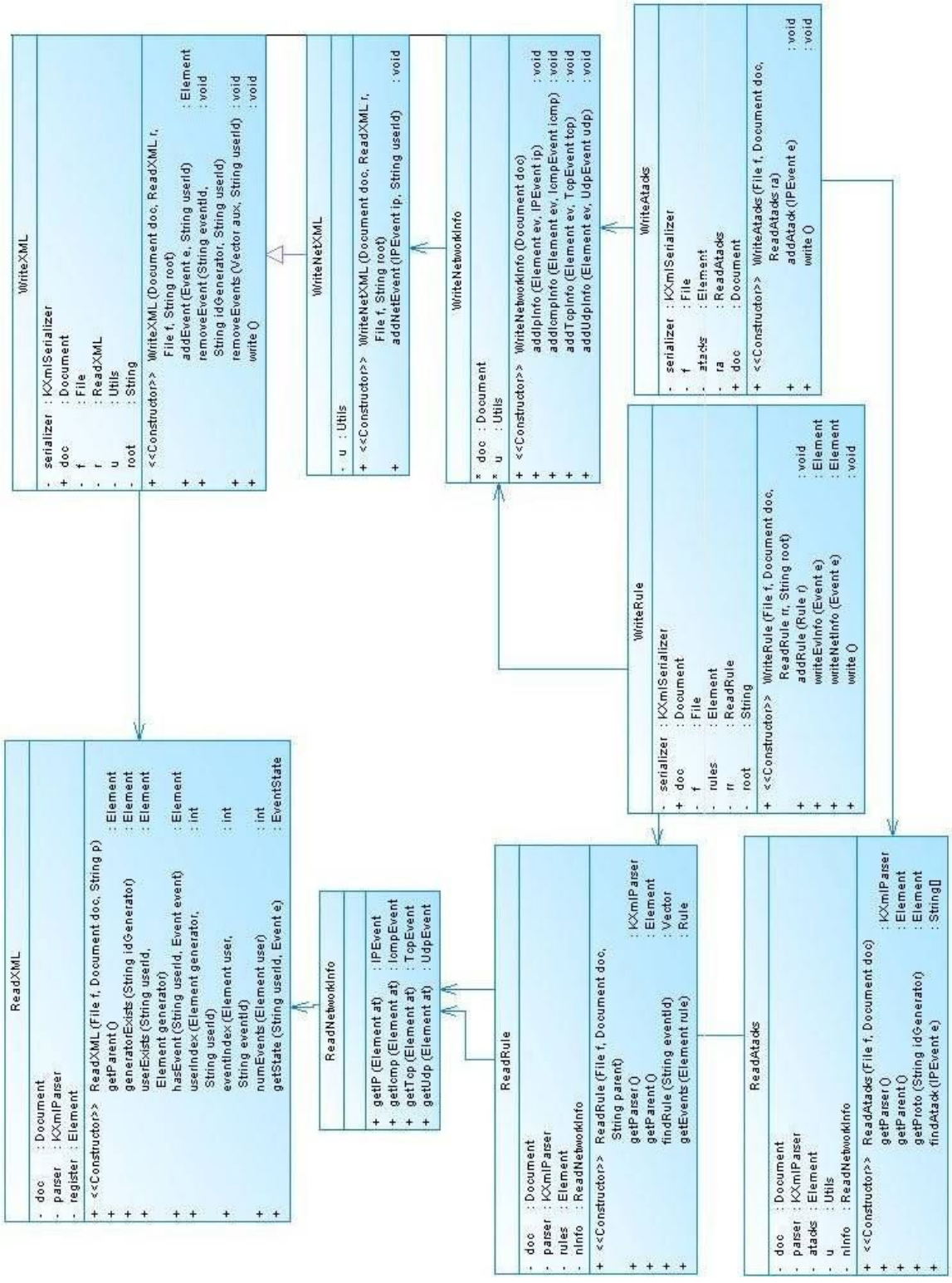


Figura 11: Diagrama UML del paquete KXML

El paquete KXML es el paquete que contiene las clases que van a permitir leer y escribir en los distintos documentos XML que guardan la información del monitor de acciones y forman su base de datos.

De este modo este paquete apoyará al administrador de eventos ya que va a permitir acceder a la información que guarda el registro de eventos de red y de aplicación.

También apoyará al motor de reglas ya que permite acceder a las reglas que se almacenan dentro del fichero de registro de reglas.

Por último también apoyará al sistema de patrones ya que gracias a este paquete se podrá acceder a los eventos que se consideran ataques potenciales y que están almacenados en un documento XML.

Como queremos conocer la información almacenada en el registro de eventos, ataques potenciales y reglas se necesita una librería que nos permita leer y escribir información en dichos ficheros. Como los ficheros de registro están codificados en XML, la librería que consideramos óptima será pues, KXML [15].

A continuación se muestran las clases de este paquete que van a permitir realizar todas las operaciones anteriormente comentadas:

❖ **Clase `ReadXML`**

Esta clase es la que permite leer la información que es común a los eventos de red y de aplicación (es decir, todos los atributos que encontramos en la clase Event) y que se almacena en el registro de aplicación o en el registro de red.

Para poder acceder a los documentos XML, la clase ReadXML utilizará la librería KXML útil para el parseo de documentos XML en dispositivos limitados.

Los atributos que tiene esta clase son los siguientes:

- **Document doc:** Documento de la librería KXML que representa al documento XML que es el registro de eventos.
- **KXmlParser parser:** parseador KXML del registro.
- **Element register:** representa al elemento raíz del documento XML.

Los métodos más importantes que presenta esta clase se muestran a continuación:

- **ReadXML (File f, Document doc, String p):** constructor. Construye un nuevo objeto ReadXML a partir de los parámetros:
    - File f: fichero que representa el documento XML que se pretende leer.
    - Document doc: documento KXML.
    - String p: cadena de texto que indica el nombre del elemento raíz del registro. Si es un registro de aplicación este String será “AplRegister”. Si, por lo contrario, es un registro de red, el String será “NetworkRegister”.
- **Element generatorExists (String idGenerator):** método que comprueba si una determinada aplicación o protocolo tiene almacenados eventos en su interior y. por tanto, se encuentra en el fichero XML.

El método recibe los siguientes parámetros:

- String idGenerator: identificador de la aplicación o del protocolo.

Si el método ha encontrado la aplicación o el protocolo devuelve un elemento XML representado por la clase Element con toda la información que contiene ese elemento, es decir, todos los usuarios que han generado eventos para la aplicación y los eventos que estos han generado.

Si no se ha encontrado la aplicación o el protocolo, se devolverá el mismo Element a null.

- **Element userExits (String userId, Element generator):** método que busca un determinado usuario dentro de una aplicación o un protocolo.
  El método recibe los siguientes parámetros:
  - String userId: identificador del usuario que se está buscando
  - Element generator: elemento XML dentro del cual buscamos al usuario.

  Si se ha encontrado al usuario, se devolverá un Element con la información XML que almacena el usuario. Si no encuentra al usuario, devolverá el Element nulo.

- **Element hasEvent (String userId, Event event):** método que comprueba si un usuario tiene registrado un evento determinado. Para poder realizar esta operación recibe los siguientes parámetros:
  - String userId: identificador del usuario del que se busca el evento
  - Event event: evento que se busca dentro del usuario.

  Si el usuario tiene el evento registrado, el método devolverá un Element que representa al evento XML registrado. Si por lo contrario, no se encuentra el evento, se devolverá Element nulo.

- **EventState getEventState (String userId, Event e):** método que busca el estado de un evento registrado por un usuario.
  Recibe los siguientes parámetros:
  - String userId: identificador del usuario que tiene registrado el evento.
  - Event e: evento del que se busca el estado.

  El método devolverá el estado del evento representado por EventState.

❖ **Clase `ReadNetworkInfo`**

Clase útil para leer la información de los eventos de red almacenados en el registro de eventos de red.

Esta clase no tiene atributos y presenta los siguientes métodos:

- **IPEvent ipEvent ():** busca un evento IP almacenado en el registro. Si lo encuentra devuelve un objeto IPEvent con la información registrada.

- **IcmpEvent icmpEvent ():** busca un evento ICMP almacenado en el registro. Si lo encuentra devuelve un objeto IcmpEvent con la información registrada.

- **TCPEvent tcpEvent ():** busca un evento TCP almacenado en el registro. Si lo encuentra devuelve un objeto TcpEvent con la información registrada.

- **UdpEvent udpEvent ():** busca un evento UDP almacenado en el registro. Si lo encuentra devuelve un objeto UdpEvent con la información registrada.

❖ **Clase `ReadRule`**

Clase que permite leer el registro XML de reglas. Contiene los siguientes atributos:

- **Document doc:** Documento KXML que representa el documento de reglas.

- **KXmlParser parser:** parseador del documento XML.

- **Element rules:** elemento raíz del documento que se lee.

- **ReadNetworkInfo ninfo:** para poder leer la información de los eventos de red que se contengan en la regla.

Entre los métodos de ReadRule destacan:

- **ReadRule (File f, Document doc, String parent):** construye un Nuevo objeto ReadRule a partir de los parámetros:

  - File f: Fichero donde se almacenan las reglas.
  - Document doc: documento que representa el fichero XML f.
  - String parent: nombre del elemento raíz.

- **Vector findRule (String eventId):** este método comprueba si un evento determinado tiene reglas. Si el evento tiene reglas, el método devuelve un Vector con todas las reglas que se han encontrado para el evento. Cada regla contendrá todos los eventos de red y aplicación que se deben cumplir para que salte la regla. Entre todos esos eventos, se debe incluir el evento que se está buscando.
  Si no tiene reglas el Vector estará vacío.
  Para saber de qué evento se busca la regla por parámetro se recibe por parámetro el identificador único del evento.

❖ **Clase `ReadAtacks`**

Clase que interactúa con el sistema de patrones de eventos de red ya que está encargada de proporcionar los métodos necesarios para poder leer el documento en el que se almacena la información de los ataques potenciales.

En esta clase aparecen los siguientes atributos:

- **Document doc:** Documento KXML que representa el documento de ataques potenciales.

- **KXmlParser parser:** parseador del documento XML de ataques.
- **Element atacks:** elemento raíz del documento XML
- **ReadNetworkInfo ninfo:** atributo que permitirá leer la información específica de red de los ataques.

Entre los métodos de esta clase destacamos los siguientes:

- **ReadAtacks (File f, Document doc):** crea un nuevo objeto ReadAtacks a partir de los parámetros:
  - File f**:** fichero donde se contienen los ataques potenciales.
  - Document doc: documento KXML que representa el fichero XML.
- **String [] findAtack (IPEvent e):** método que comprueba si un evento es ataque potencial o no. Para ello busca el evento en el registro: si el evento aparece en el registro se considera ataque potencial y se devuelve un array de String en el que la primera posición es ocupada por el identificador del evento y la segunda la ocupa la descripción del evento que aparece en el registro de ataques.
  Si el evento no aparece en el documento, el ataque no se considera como potencial y se devuelve un array con Strings nulos.
  El evento que se desea buscar es recibido por parámetros:

❖ **Clase `WriteXML`**

Clase que ofrecerá los métodos con los que se podrá modificar el fichero de registro de eventos de aplicación o registro de eventos de red. Gracias a esta clase se podrá modificar o escribir todos aquellos datos que son comunes para eventos de red y aplicación.

Para poder realizar esta labor es necesario utilizar la librería java KXML.

Esta clase presenta los siguientes atributos:

- **KXmlSerializer serializer:** serializador que permite escribir en el fichero.
- **Document doc:** clase que representa al documento XML que se pretende escribir.
- **File f:** fichero que se modifica.
- **ReadXML r:** será necesaria esta clase para poder leer el documento.
- **String root:** nombre del elemento raíz del documento a escribir.

A continuación se presentan los métodos que van a permitir realizar modificaciones en el registro de eventos:

- **WriteXML (Document doc, ReadXML r, File f, String root):** crea un nuevo objeto WriteXML a partir de los parámetros:
  - Document doc: representación del documento XML.
  - ReadXML r: lector del documento.
  - File f: fichero en el que se escribe.

- String root: elemento raíz.

- **Element addEvent (Event e, String userId):** añade un evento a un usuario. El evento se añadirá con toda la información que éste tenga registrada. Como resultado se devolverá el elemento Element KXML añadido. Será nulo si el evento no se ha podido añadir.
  El método recibe los siguientes parámetros:

  - Event e: evento que se va escribir en el registro.
  - String userId: identificador del usuario al que se registra el evento.

- **void removeEvent (String eventId, String idGenerator, String userId):** método que borra un evento a un usuario. Es decir, se elimina el evento del registro.
  Para saber qué evento se borra y a quién se utilizan los parámetros:

  - String eventId: identificador del evento que se elimina.
  - String idGenerator: identificador de la aplicación o protocolo que generó el evento.
  - String userId: identificador del usuario al que se borra el evento.

❖ **Clase `WriteNetXML`**

Hereda de WriteXML ya que permite realizar las modificaciones al registro de eventos para los campos comunes a los eventos de red y aplicación y, además, permite modificar aquellos atributos que son específicos para los eventos de red.

WriteNetXML sólo incluye un método más que se muestra a continuación:

- **void addNetEvent (IPEvent e, String userId):** añade un evento de red al usuario indicado por parámetros:

  - IPEvent e: evento de red que se añade.
  - String userId: identificador del usuario al que se añade el evento.

❖ **Clase `WriteNetworkInfo`**

Clase que se encarga de escribir en el documento XML los campos específicos para cada protocolo. Se crean 4 métodos, uno por protocolo, en el que cada método se escribe en el documento XML los campos que definen a ese protocolo. Así tendremos los métodos:

- **Void addIpInfo (Element ev, IPEvent ip):** añade las cualidades del protocolo IP.

- **Void addIcmpInfo (Element ev, IcmpEvent icmp):** añade los campos del protocolo ICMP.

- **Void addTcpInfo (Element ev, TcpEvent tcp):** añade las cualidades del protocolo TCP.

- **Void addUdpInfo (Element ev, UdpEvent udp):** añade las características del protocolo UDP.

❖ **Clase `WriteRule`**

Clase que permite añadir nuevas reglas al registro de reglas.

Esta clase cuenta con los siguientes atributos:

- **KXmlSerializer serializer:** para poder modificar el documento.
- **Document doc:** representa al documento XML que se modifica.
- **Element rules:** elemento raíz del registro de reglas.
- **ReadRule rr:** se necesitará leer información de las reglas ya almacenadas.

Para poder añadir reglas esta clase añade el método addRule:

- **void addRule (Rule r):** método que escribe una nueva regla en el registro de reglas incluyendo todos sus eventos de red y aplicación. La regla a escribir es la que se recibe como parámetro.

❖ **Clase `WriteAtacks`**

Clase que ofrece los métodos necesarios para escribir nuevos ataques potenciales en el registro de patrones de eventos.

En esta clase podemos encontrar los siguientes atributos:

- **KXmlSerializer serializer:** para poder modificar el documento.
- **Document doc:** representa al documento XML en el que se escribe.
- **Element atacks:** elemento raíz del documento de patrones.
- **ReadAtacks ra:** se necesitará leer información del documento.

Se conseguirá añadir nuevas reglas gracias al método addAtack:

- **void addAtack (IPEvent e):** añade un nuevo ataque potencial al sistema de patrones. El evento que se debe añadir y, a partir de ahora, es el evento IP que se pasa por parámetro.

4.1.4 Paquete sniffer

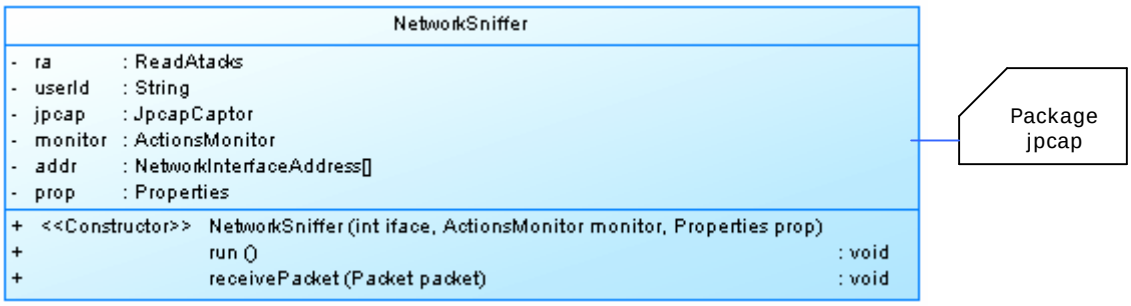


Figura 12: Diagrama UML del paquete sniffer

El paquete sniffer representa al sensor de eventos de aplicación. Su función es la de escuchar todo el tráfico de red que entra y sale del dispositivo. Si las tramas que llegan al dispositivo vienen encapsuladas en los protocolos TCP, UDP o ICMP, construye un objeto que represente el paquete y lo transmite al sistema de patrones para que éste compruebe si el paquete puede considerarse como un ataque potencial o no.

Para poder escuchar todo el tráfico de red que llega al dispositivo es necesario utilizar las clases que nos proporciona la librería JPCAP [16].

Este paquete contiene una única clase que es la que se encarga de de realizar las tareas anteriormente especificadas:

❖ **Clase `NetworkSniffer`**

Hilo Java que emula a un Sniffer de red que lee el tráfico de Internet que llega a través de un interfaz de red del dispositivo. Para ello, se apoya de la librería Java Jpcap [16].

Esta clase posee los siguientes atributos:

- **ReadAtacks ra:** objeto ReadAtacks para poder leer el archivo XML que contiene registrados los ataques potenciales.
- **String userId:** identificador del usuario.
- **JpcapCaptor jpcap:** objeto de la librería Jpcap que permite escuchar el interfaz de red.
- **ActionsMonitor monitor:** monitor de acciones al que irán los paquetes que se consideren ataques potenciales.
- **NetworkInterfaceAddress[] addr:** array en el que se almacenan las direcciones IP de los interfaces de red disponibles en el dispositivo
- **Properties prop:** objeto para poder saber la ruta en la que se encuentra el fichero de ataques potenciales.

Los métodos que tiene NetworkSniffer son los siguientes:

- **NetworkSniffer(int iface, ActionsMonitor monitor, Properties prop):** crea un nuevo objeto NetworkSniffer a partir de los siguientes parámetros:
    - int iface: número del interfaz de red por el que se escucha.
    - ActionsMonitor monitor: monitor de acciones de la aplicación.
    - Properties prop: fichero de propiedades.
- **ReceivePaquet (Packet packet):** método que recibe un paquete que llega o sale del interfaz de red por el que se escucha. Sólo recibe paquetes IP debido a un filtro que se aplica. Analiza el paquete y lo transmite al sistema de patrones.
  Este método recibe un único parámetro:
    - Packet packet: paquete que llega o sale al interfaz de red.

4.1.5 Paquete wsfep

Paquete que contiene la aplicación software Java ME que se pretende monitorizar. Esta aplicación recibe el nombre de WSFEP [17].

WSFEP es una aplicación desarrollada como un proyecto de fin de carrera dentro del grupo de computación ubicua del grupo de Aplicaciones y Servicios Telemáticos (GAST) del departamento de Ingeniería Telemática de la universidad Carlos III de Madrid.

La aplicación es una aplicación que comparte ficheros entre dos o más dispositivos. Cada dispositivo se une a un grupo multicast gracias al cual puede ver los ficheros que se encuentran disponibles en todos los dispositivos. El dispositivo que utilice esta aplicación podrá descargar los ficheros disponibles en la red que están compartidos.

WSFEP crea su propio protocolo de transferencia de ficheros entre dispositivos limitados conectados a la misma red. Para ello, se basa en los protocolos eMule Protocol [21] y Pervasive Discovery Protocol [22].

La aplicación funciona de tal modo que los dispositivos que forman una redad-hoc pueden conocer todos los ficheros que comparten el resto de dispositivos y los ficheros que el propio dispositivo tiene a disposición del resto de usuarios. Si un usuario decide descargar un fichero que se encuentre en remoto, WSFEP le ofrecerá la posibilidad de una descarga eficiente y segura. La seguridad está basada en un sistema de gestión de confianza denominado PTM (*Pervasive Trust Management Model*).

Cada acción que cualquier usuario realice en el dispositivo que corre la aplicación es logueada en un fichero de log específico para dicho dispositivo. Este fichero tendrá el mismo formato que el especificado para el módulo sensor de eventos de aplicación.

Cada vez que se loguee un evento en el fichero de log de WSFEP se notificará al monitor de acciones para que este decida o no si el evento es un ataque.

4.1.6 Paquete utilidades

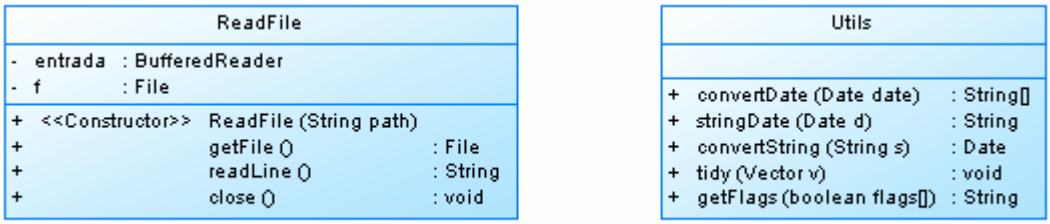


Figura 13: Diagrama UML del paquete utilidades

El paquete utilidades contiene las utilidades del sistema. Contiene clases que no representan a ningún módulo pero que son necesarias para realizar ciertas operaciones básicas necesarias dentro del conjunto global de la aplicación.

Este paquete incluye dos clases que no presentan ninguna relación entre ellas. A continuación mostramos esas clases:

❖ **Clase `ReadFile`**

Clase que nos permite leer el fichero de texto en el que se almacenan los eventos de la aplicación WSFEP monitorizada.

Esta clase cuenta con los siguientes atributos:

- **BufferedReader entrada:** buffer para leer el fichero.
- **File f:** fichero que se lee.

Para poder leer el fichero de log de la aplicación se apoya en los siguientes métodos:

- **ReadFile (String path):** Construye un nuevo objeto ReadFile a partir del path en el que se éste se encuentra. El path del fichero es recibido por parámetro.
- **File getFile ():** método que devuelve el fichero que se está leyendo.
- **String readLine ():** método que lee una línea del fichero. Se lee la línea que sigue a la última línea recibida. Este método devuelve la cadena de texto leída.

❖ **Clase `Utilities`**

Clase que contiene varios métodos útiles para el manejo de fechas y arrays de flags.

Los métodos que contiene esta clase son los siguientes:

- **String [] convertDate (Date date):** devuelve un array de String que representa el objeto Date introducido como parámetro según unos patrones yyyy-MM-dd para la fecha y hh:mm:ss para la hora.
  El parámetro que recibe este método es el siguiente:

- Date date: fecha que se quiere convertir a String

El método devolverá un Array de String en el que la posición 0 tendrá guardada la fecha y en la posición 1 la hora.

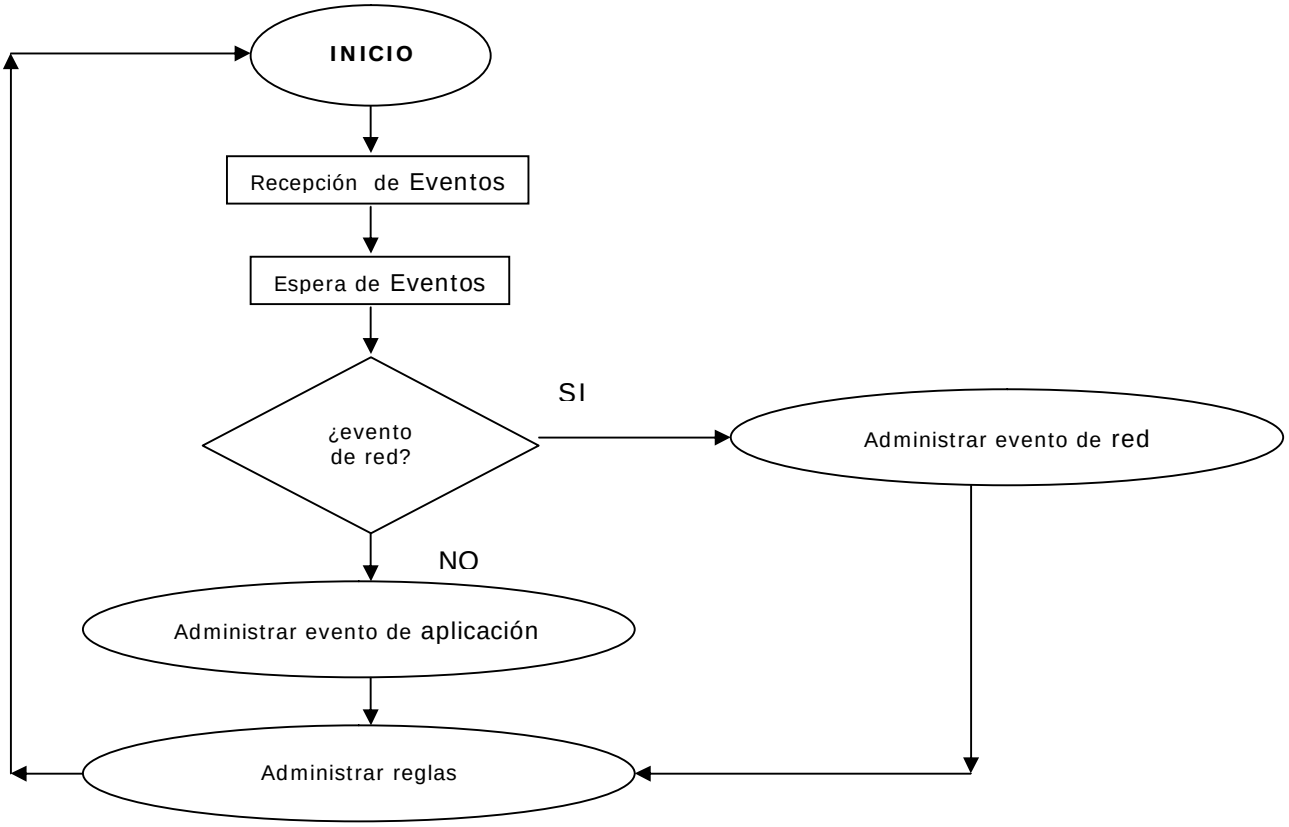
- **Date converString (String s):** método que realiza la operación opuesta a la que realiza el método anterior. Convierte un String que tiene una fecha y hora escrita y lo convierte a un objeto Date que devuelve. Por parámetro recibe el String que se desea transformar en fecha.

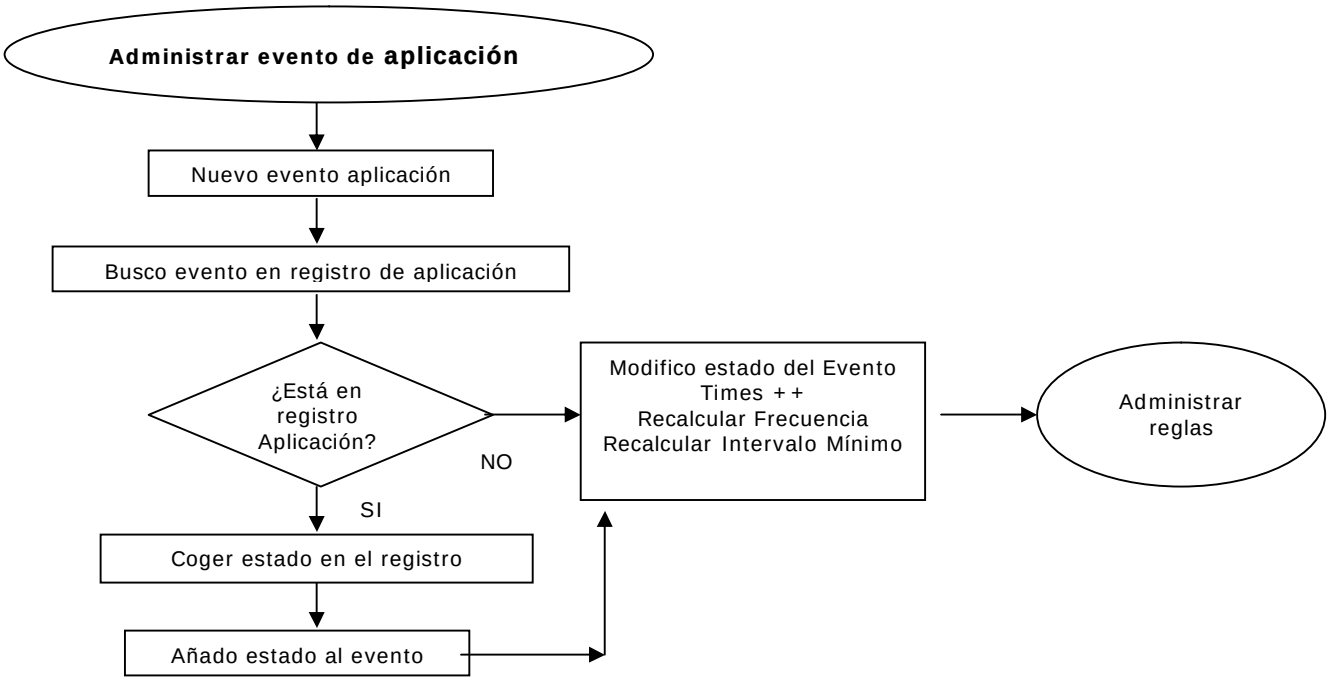
- **Tidy (Vector v):** método que ordena un Vector que contiene una regla en cada posición. Las reglas se ordenan en función de su campo prioridad, de menor prioridad a mayor prioridad.
  Por parámetro se recibe el Vector de reglas que se debe ordenar

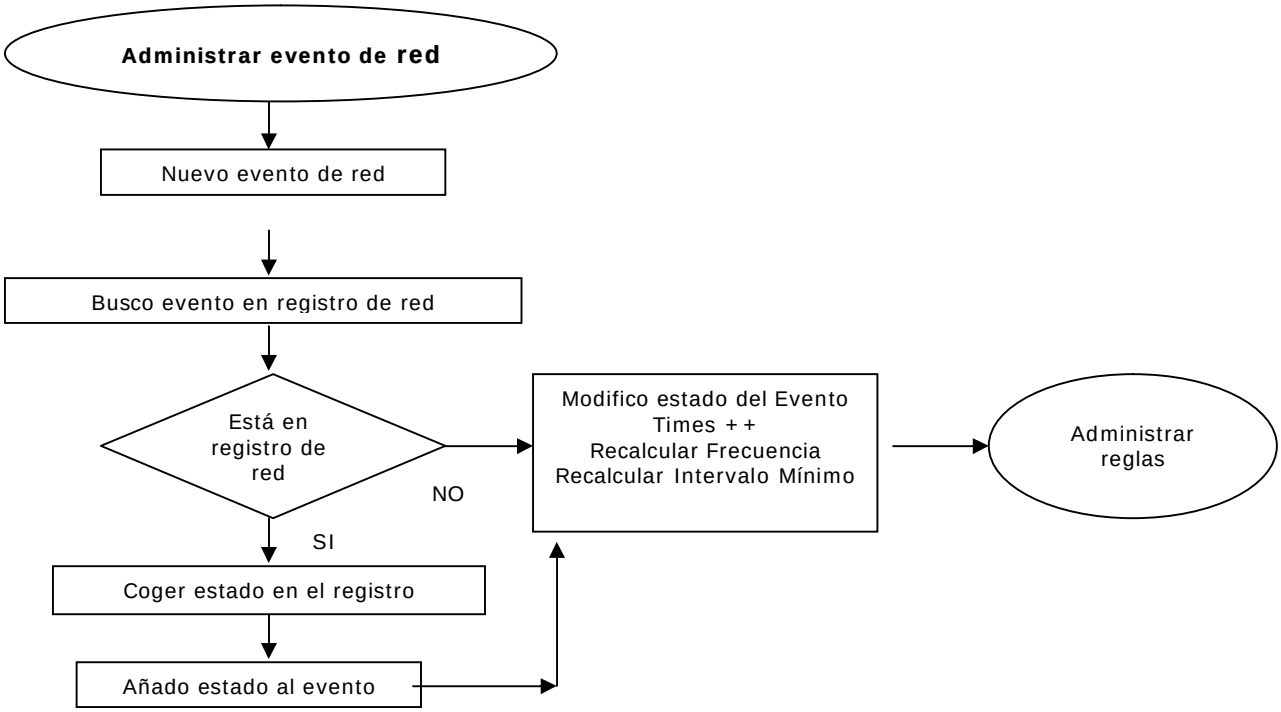
- **String getFlags (boolean flags []):** método que analiza una array que representa los flags de protocolo IP o TCP. Obtiene el valor de cada flag, si ese valor es verdadero añade un 1 a un String, si es falso añade un 0. Se devolverá el String creado que representa los flags.
  Este método recibe por parámetros en array de flags a convertir a String.

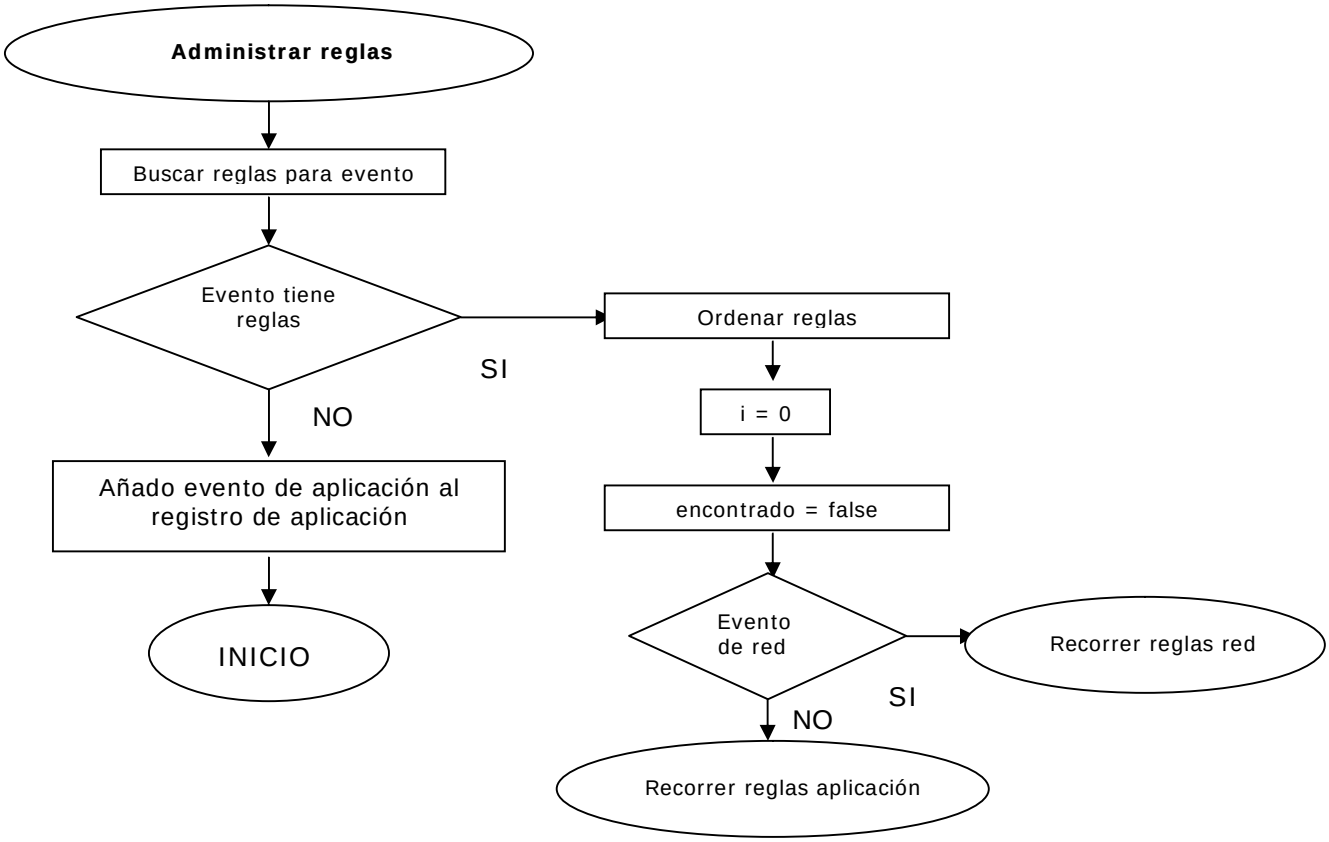
4.2 Diagrama de Flujo

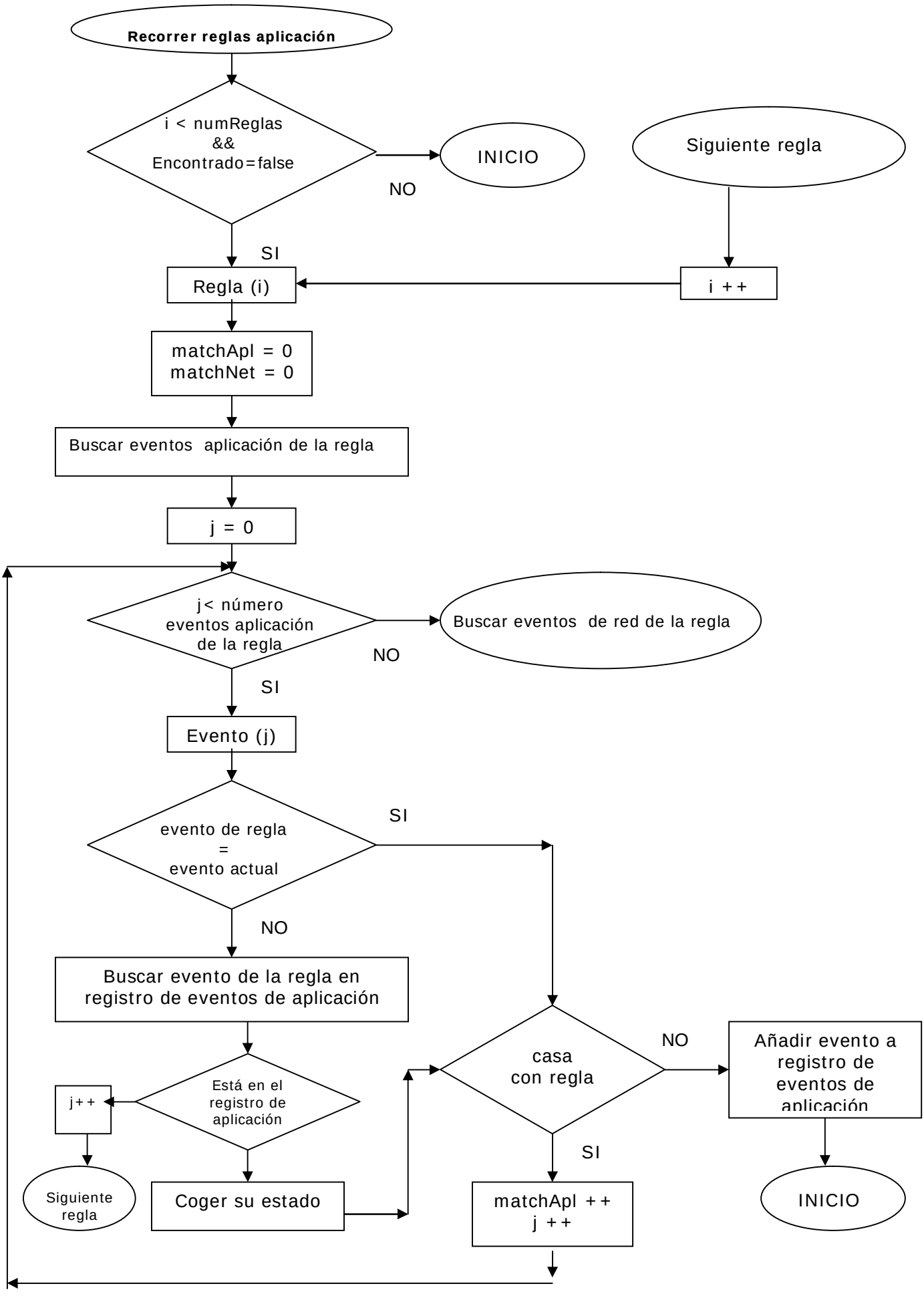
En este apartado se presenta el diagrama de flujo del hilo principal de la aplicación dividido por funciones:

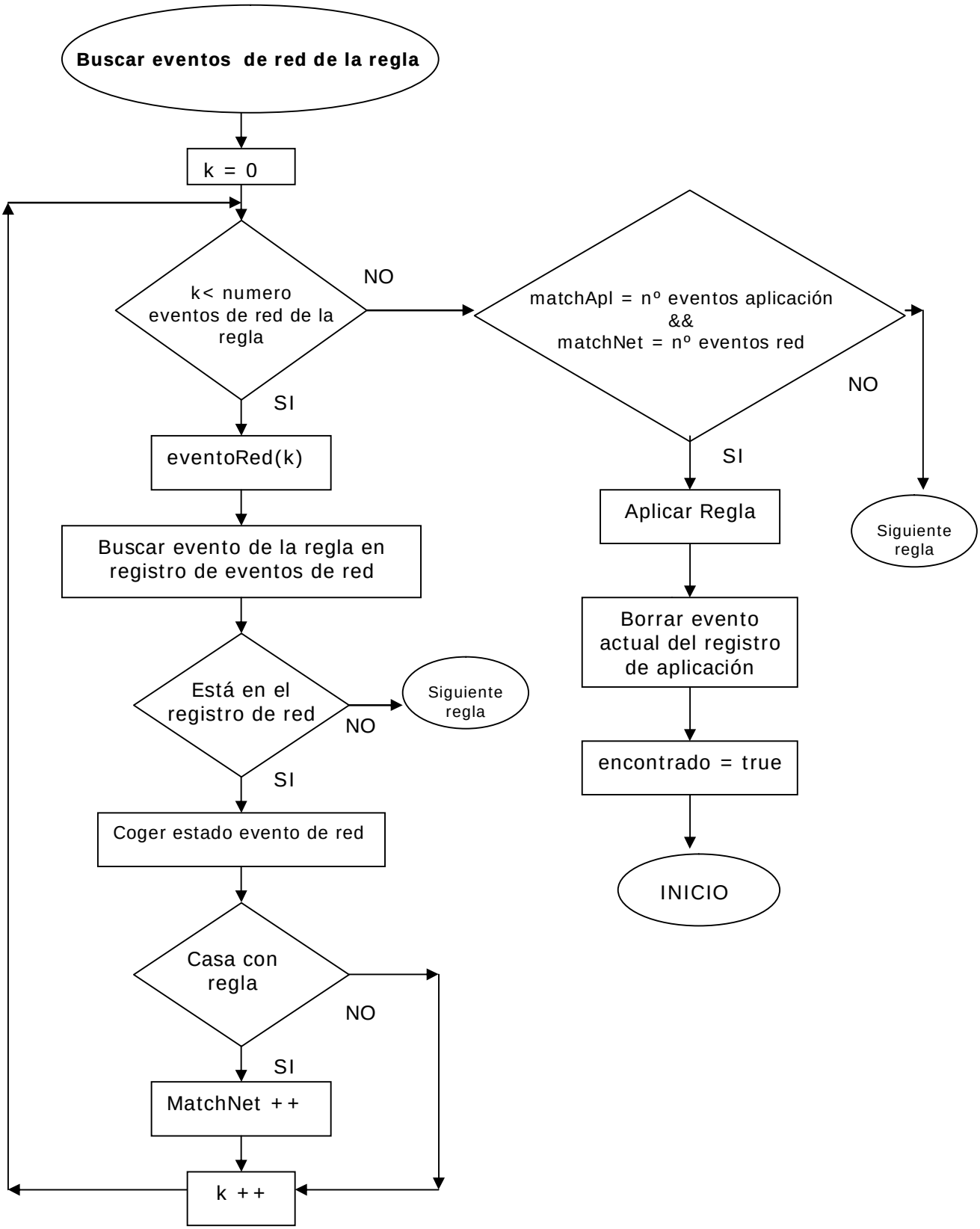


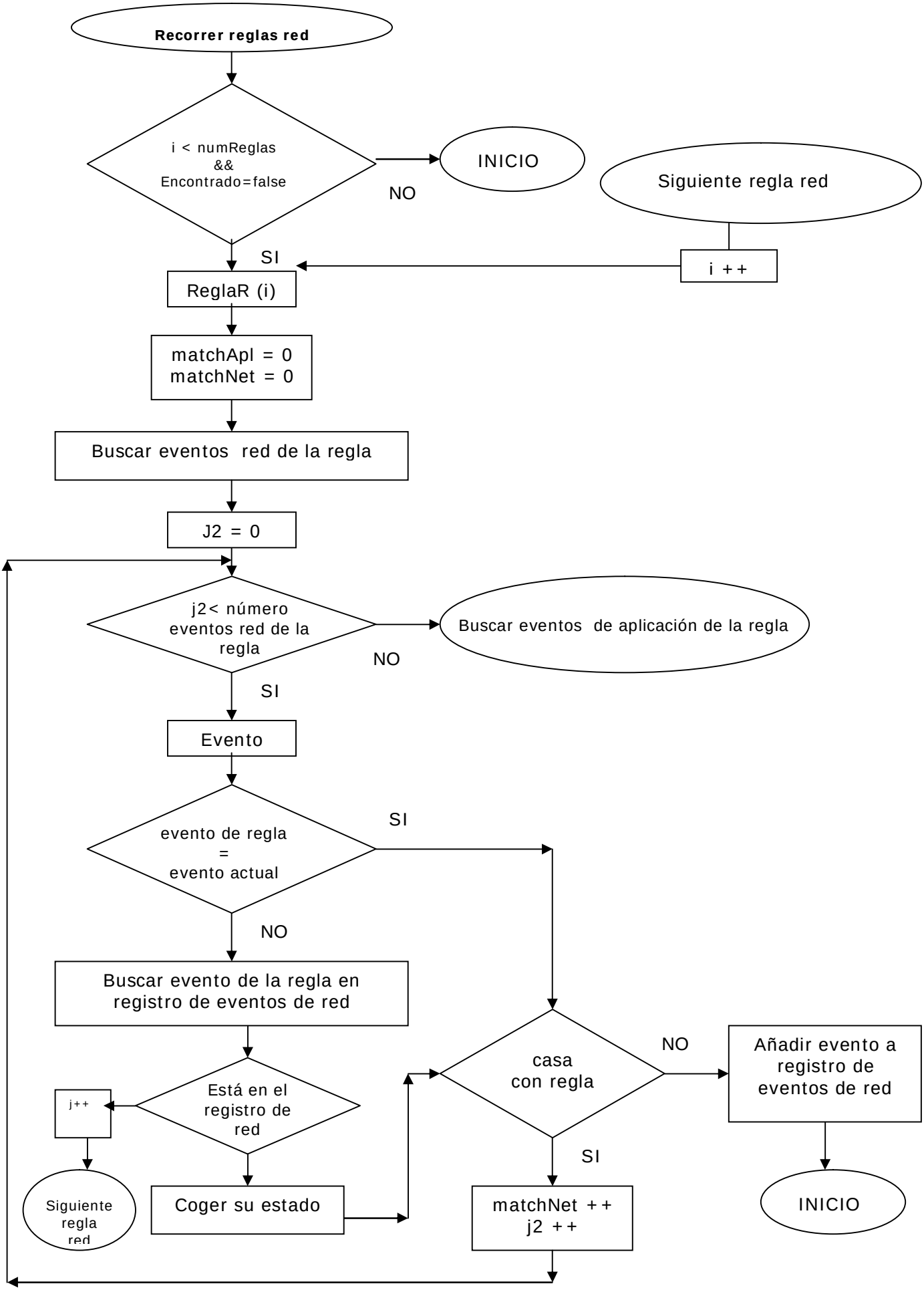


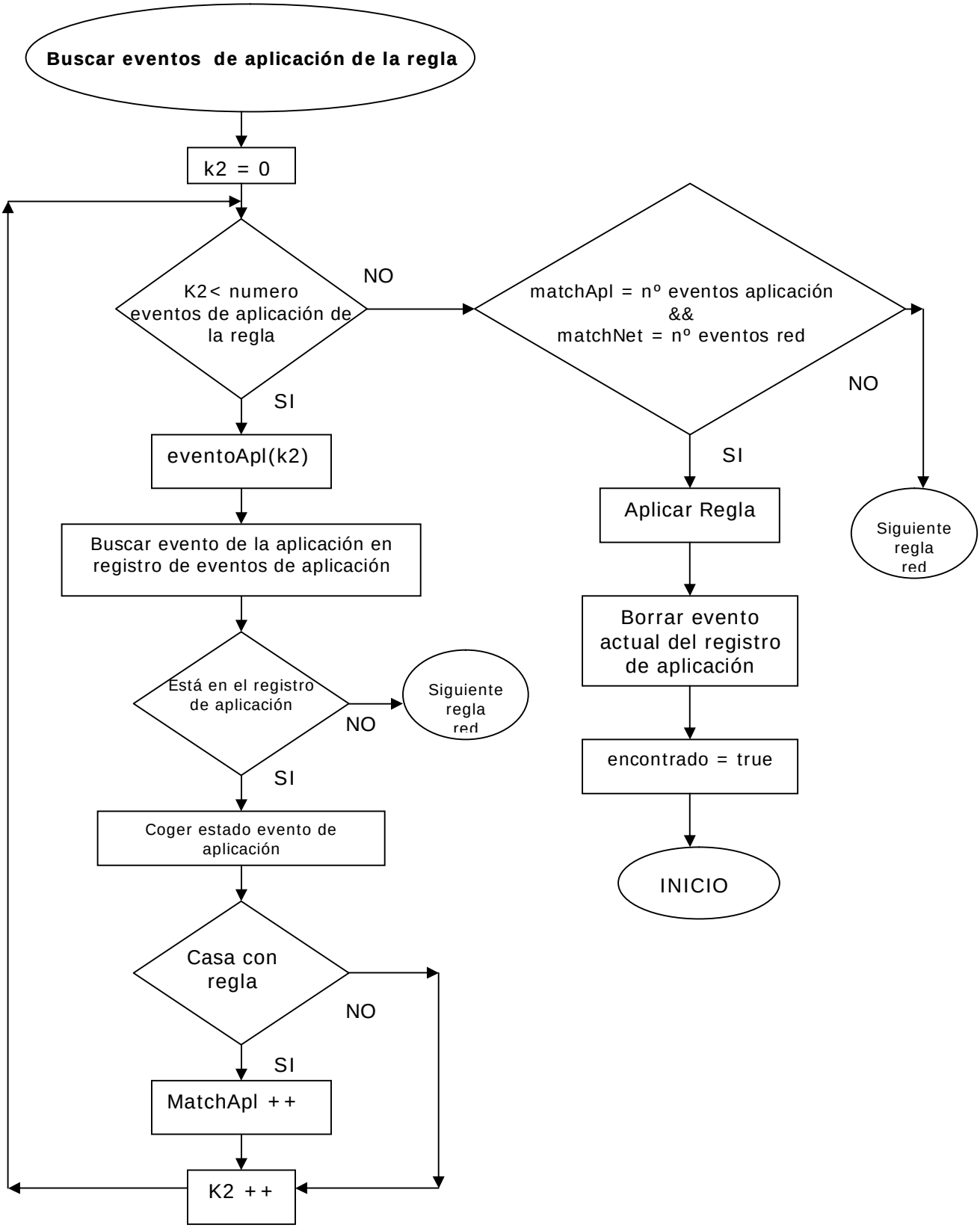

Administrar evento de red
Nuevo evento de red
Busco evento en registro de red
Está en registro de red
NO
SI
Modifico estado del Evento
Times + +
Recalcular Frecuencia
Recalcular Intervalo Mínimo
Administrar reglas
Coger estado en el registro
Añado estado al evento


Administrar reglas
Buscar reglas para evento
Evento tiene reglas
SI
NO
Ordenar reglas
i = 0
encontrado = false
Evento de red
SI
NO
Recorrer reglas red
Recorrer reglas aplicación
Añado evento de aplicación al registro de aplicación
INICIO


Recorrer reglas aplicación
i < numReglas && Encontrado=false
NO
INICIO
Siguiente regla
SI
Regla (i)
i ++
matchApl = 0
matchNet = 0
Buscar eventos aplicación de la regla
j = 0
j< número eventos aplicación de la regla
NO
Buscar eventos de red de la regla
SI
Evento (j)
evento de regla = evento actual
SI
NO
Buscar evento de la regla en registro de eventos de aplicación
Está en el registro de aplicación
j++
Siguiente regla
Coger su estado
casa con regla
NO
Añadir evento a registro de eventos de aplicación
SI
matchApl ++
j ++
INICIO


Buscar eventos de red de la regla
k = 0
k< numero eventos de red de la regla
NO
SI
eventoRed(k)
Buscar evento de la regla en registro de eventos de red
Está en el registro de red
NO
Siguiente regla
SI
Coger estado evento de red
Casa con regla
NO
SI
MatchNet ++
k ++
matchApl = nº eventos aplicación && matchNet = nº eventos red
NO
SI
Aplicar Regla
Siguiente regla
Borrar evento actual del registro de aplicación
encontrado = true
INICIO


Recorrer reglas red
i < numReglas && Encontrado=false
NO
INICIO
Siguiente regla red
i ++
SI
ReglaR (i)
matchApl = 0
matchNet = 0
Buscar eventos red de la regla
J2 = 0
j2< número eventos red de la regla
NO
Buscar eventos de aplicación de la regla
SI
Evento
evento de regla = evento actual
SI
NO
Buscar evento de la regla en registro de eventos de red
Está en el registro de red
j++
Siguiente regla red
Coger su estado
casa con regla
NO
Añadir evento a registro de eventos de red
SI
matchNet ++
j2 ++
INICIO


Buscar eventos de aplicación de la regla
k2 = 0
K2< numero eventos de aplicación de la regla
NO
SI
eventoApl(k2)
Buscar evento de la aplicación en registro de eventos de aplicación
Está en el registro de aplicación
NO
Siguiente regla red
SI
Coger estado evento de aplicación
Casa con regla
NO
SI
MatchApl ++
K2 ++
matchApl = nº eventos aplicación
&&
matchNet = nº eventos red
NO
SI
Aplicar Regla
Siguiente regla red
Borrar evento actual del registro de aplicación
encontrado = true
INICIO

Capítulo

5

5 Interfaz gráfica.

El monitor de acciones se presenta al usuario a través de una interfaz gráfica muy sencilla para que, gracias a ella, el usuario observe los resultados de la monitorización y configure las reglas y los ataques que desee.

A continuación se muestra el aspecto gráfico y funcional de la aplicación de modo que pueda servir de presentación de la aplicación.

Cuando se arranca la aplicación aparece la siguiente pantalla de inicio:

Figura 14: Pantalla de Inicio del monitor de acciones

Como se muestra en la figura 14 el monitor de acciones cuando arranca nos ofrece tres opciones: “Comenzar”, “Configuración”, “Últimos Resultados”:

- **Comenzar:** orientará al usuario al comenzar la monitorización de su sistema.
- **Configuración:** guía al usuario a configurar el monitor de acciones permitiendo añadir nuevas reglas o ataques potenciales.
- **Últimos Resultados:** muestra al usuario los últimos resultados obtenidos en sesiones de monitorización anteriores.

5.1 Inicio de la monitorización

Para poder iniciar la monitorización del sistema será necesario pulsar en el botón "comenzar" de la pantalla de inicio de monitor mostrada en la figura 14. Cuando se haya pulsado dicho botón, aparecerá la siguiente pantalla:

Figura 15: Pantalla de inicio de sesión monitor de acciones

La pantalla que se muestra en la figura 15 permite al usuario configurar el inicio de sesión, indicando si se desea iniciar una nueva sesión o no o si desea incluir o no Network Sniffer:

- **Iniciar nueva sesión:** si el usuario decide iniciar una nueva sesión del monitor de acciones, no tendrá en cuenta el contenido del registro de eventos de monitorizaciones anteriores (es decir, se elimina el contenido de este registro), y creará un nuevo fichero de log en el que se loguearán todos los ataques que se produzcan para esta nueva sesión. El fichero de log se nombrará de la siguiente forma: "aaaammdd hh-mm.monitor.log", donde aaammdd hh-mm es la fecha y hora en la que se inicia la sesión.
  Si el usuario no desea iniciar una nueva sesión, se tendrán en cuenta los últimos eventos almacenados en el registro de red y se loguearán los eventos en el último evento de log creado. De este modo, se continúa con la última sesión del monitor ejecutada.

- **Incluir Network Sniffer:** el usuario podrá elegir si desea o no incluir la monitorización del interfaz de red activando o no esta opción. Es decir, activando esta opción se tendrán en cuenta todos los eventos que llegan a través de Internet, desactivándola, los eventos de Internet se ignoran.
  En el caso de que el usuario desee incluir la monitorización del interfaz de red, podrá indicar cuál es el interfaz que desea monitorizar gracias a la lista desplegable que aparece señalizada como "Nº Interfaz". En esta lista aparecerán todos los interfaces que están disponibles en el dispositivo.

El usuario deberá indicar obligatoriamente la configuración que desea para la sesión del monitor de acciones. Si no indicara los campos de configuración, se mostrarán los siguientes errores:

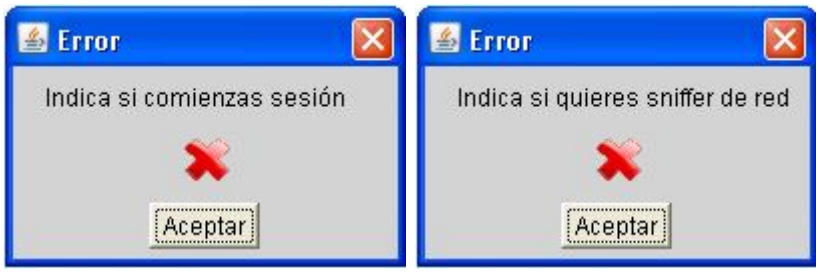


Figura 16: Errores

Tras haber indicado los parámetros de inicio de sesión adecuadamente se muestra la pantalla principal del monitor de acciones:

Figura 17: Sesión del monitor de acciones.

En la figura 17 se muestra una sesión del monitor de acciones. Vemos que la pantalla se divide en dos secciones principales: un campo de texto y una línea de botones.

En el campo de texto aparecen todos los ataques que el monitor ha detectado escritos en el mismo formato que presentan en el fichero de log. Si el usuario ha indicado que desea una nueva sesión, esta área de texto aparecerá vacía, si no por lo contrario se está continuando una sesión anterior, se mostrarán todos los eventos logueados de dicha sesión.

Los botones que aparecen en el monitor son los siguientes:

- **Detener:** detiene momentáneamente la monitorización. Es decir, no se tendrán en cuenta todos los eventos que lleguen al monitor. Cuando se acciones este botón, se cambiará su texto a “iniciar”, de modo que cuando se active de nuevo el botón, se volverá a iniciar la monitorización de los eventos de red (si esta opción está activada) y de aplicación.

- **Salir:** cierra el monitor de acciones saliendo de la aplicación.

Los ataques detectados por el monitor de acciones, como ya se ha visto, se muestran en la pantalla principal de la figura 17. Si además, la acción de la regla

que aplica el evento es “log/msg”, se mostrará una ventana de diálogo con la notificación del evento para que así el usuario descubra el ataque de una forma más visual:

Figura 18: Alertas de Ataque.

En la figura 17 se observa que el monitor de acciones aparece una barra de menú en la que se presentan dos submenús: “menú” y “Ayuda”.

Al pulsar “menú” aparecerán las mismas opciones que aparecen en los botones de la pantalla principal.

Figura 19: Menú del monitor de acciones.

Al seleccionar el menú “Ayuda” se puede acceder a la ayuda del monitor de acciones:

Figura 20: Menú Ayuda de monitor de acciones.

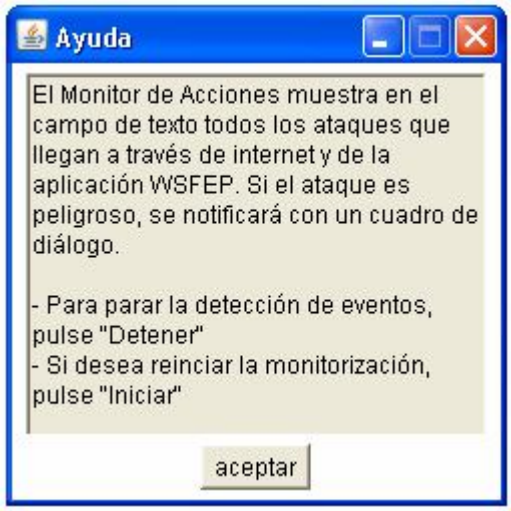


Figura 21: Ayuda de monitor de acciones.

5.2 Configuración del monitor

Podemos configurar el monitor de acciones por dos vías: añadiendo una regla y/o añadiendo un ataque potencial de red.

La ventana que nos va a permitir acceder a las opciones anteriores es la que se muestra a continuación:

Figura 22: Ventana Configuración del monitor de acciones.

Si elegimos el botón "Añadir Regla" tendremos la opción de añadir una nueva regla al sistema de reglas del monitor de acciones. Si por lo contrario, hacemos clic en el botón "Añadir Ataque Potencial de Red", se podrá añadir un nuevo patrón de ataque potencial de red.

AÑADIR UNA REGLA:

Cuando elegimos añadir regla se mostrará una ventana con la que se muestra a continuación:

Figura 23: Ventana Añadir Regla.

En la ventana Añadir Regla se pide al usuario que introduzca los siguientes datos:

- **action:** acción a realizar el evento supera los umbrales de la regla. Se pueden ejecutar tres acciones: log (se añade el ataque en el fichero de log), msg (se muestra una alerta al usuario) y log/msg (se realizan las dos opciones anteriores).

- **mensaje:** mensaje que se muestra al usuario y se escribe en el log cuando se ha producido un ataque.

- **prioridad:** número entero que indica la prioridad de la regla. La regla más prioritaria es la que tiene este valor menor.

En la ventana se presentan los siguientes botones:

- **atrás:** botón para volver a la ventana anterior de configuración.

- **cancelar:** botón para cancelar la operación que se está realizando.

- **Añadir Evento:** es el botón que se ha de pulsar una vez se han rellenado todos los datos de la regla para así introducir los datos del/de los evento/s que se incluyen en la regla. Para realizar esta operación se muestra la siguiente ventana:

Figura 24: Ventana Añadir Evento

Para poder añadir un evento a una regla es necesario especificar lo siguientes campos que son obligatorios:

- **Identificador del Evento:** es la cadena de texto que identifica al evento.

- **Descripción del Evento:** texto que describe al evento.

- **Fuente del evento:** indica quién puede generar el ataque. Podemos elegir entre la aplicación monitorizada (Wsfep) y los protocolos monitorizados (ip, icmp, tcp y udp). Dependiendo de la fuente de datos que se elija aparecerá un nuevo panel para introducir datos dependientes de la fuente.

Si la fuente de datos seleccionada es Wsfep:

Se mostrará en pantalla la siguiente ventana:

Figura 25: Ventana Añadir Evento Wsfep

Ahora es necesario rellenar los siguientes campos:

- **Regla de frecuencia / Regla intervalo:** Se indica si se tiene en cuenta la frecuencia del evento o el mínimo intervalo de tiempo entre dos eventos.

- **Número de ocurrencias:** número máximo de veces que se puede producir el evento para que no se considere ataque.

- **Intervalo de tiempo:** intervalo de tiempo en el que se mide la frecuencia del evento (si es regla de frecuencia) o mínimo intervalo de tiempo que el evento no debe superar para no considerarse ataque (si es regla de mínimo intervalo).

En la ventana aparecen los siguientes botones, comunes para todos los paneles de las diferentes fuentes de eventos:

- **Añadir otro evento:** da la opción de añadir otro evento si la regla es combinada. Se muestra una ventana igual a la actual con todos los campos vacios para añadir la información del nuevo evento.

- **Finalizar:** al pulsar este botón se añade la nueva regla al sistema de reglas del monitor de acciones.

Si la fuente de datos seleccionada es IP:

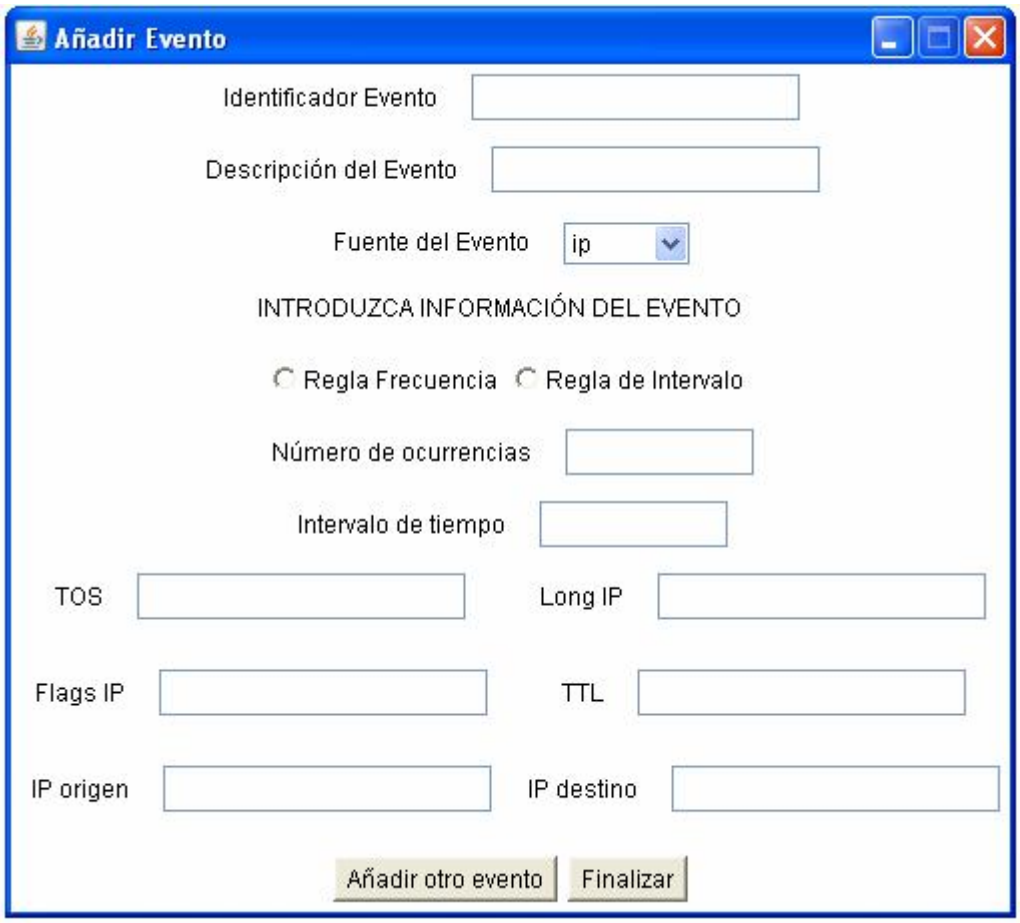


Figura 26: Ventana Añadir Evento IP.

Vemos que hay que rellenar campos comunes con Wsfep como tipo de regla, número de ocurrencias e intervalo. A continuación se definen los campos propios de un evento IP:

- **TOS:** tipo de servicio del datagrama IP.
- **Long IP:** longitud del datagrama IP.
- **Flags IP:** cadena de unos y ceros que representan los flags del datagrama IP. Estos flags son DF (don't fragment), MF (more fragment). Si por ejemplo el flag DF es cero y MF es uno, escribiríamos "01"
- **TTL:** tiempo de vida del datagrama o número máximo de saltos.
- **IP origen:** dirección IP origen del datagrama IP en formato numérico.
- **IP destino:** dirección IP destino del datagrama IP en formato numérico.

Todos los campos deben ir rellenados. Si en algún momento se quiere obviar un campo dentro de la regla, este campo se rellenará con ceros. Para ignorar el campo flags IP se debe rellenar con "000".

Si la fuente de datos seleccionada es ICMP:

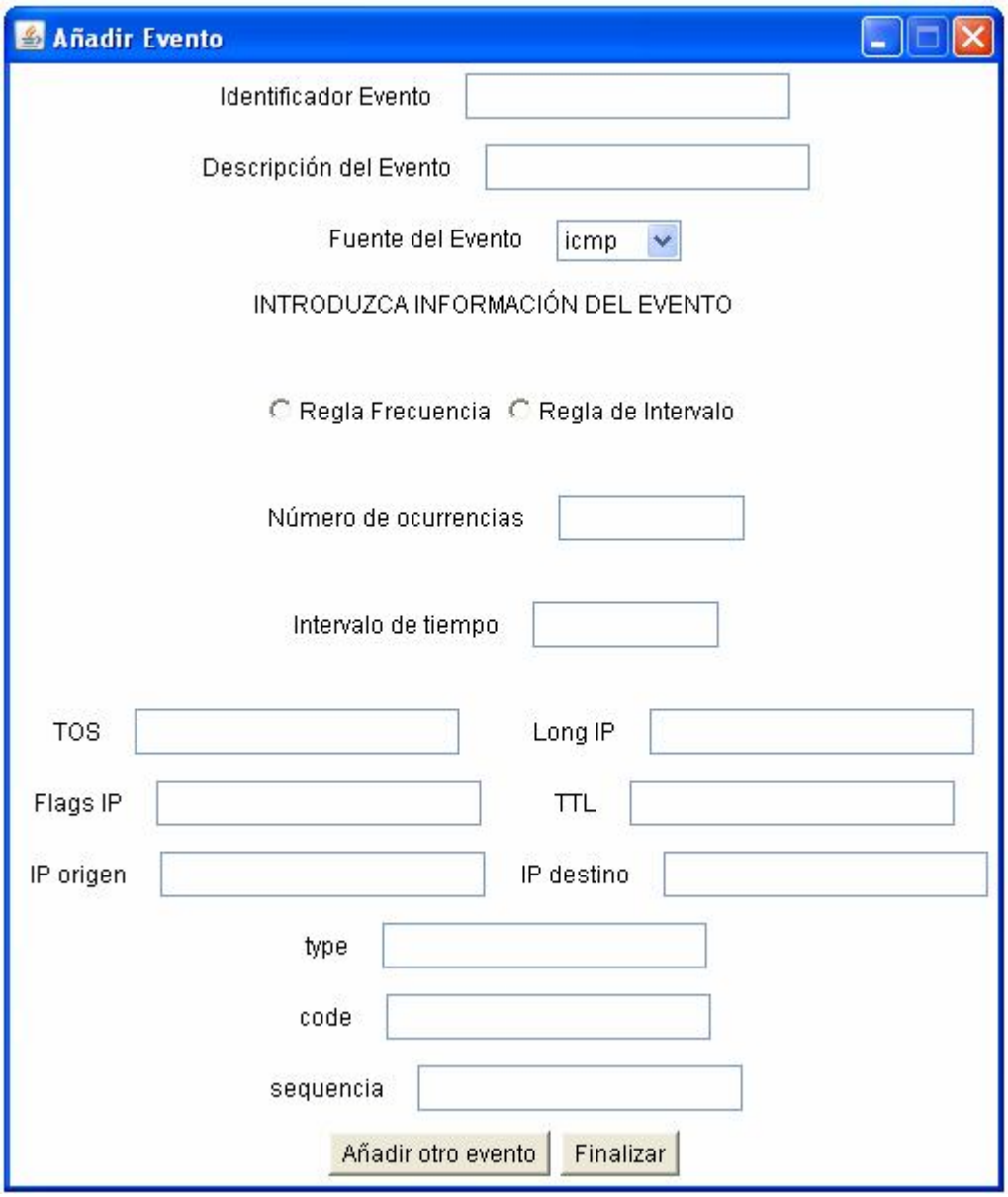


Figura 27: Ventana Añadir Evento ICMP.

Como en el caso anterior hay campos comunes con WSFEP y además hay campos comunes con IP (ya que un datagrama ICMP se encapsula dentro de uno IP).

Veamos los campos nuevos que aparecen para ICMP:

- **Type:** tipo de paquete ICMP.
- **Code:** código del paquete ICMP.
- **Secuencia:** número de secuencia del paquete.

Los campos irán con ceros si quiere que se omitan.

Si la fuente de datos seleccionada es TCP:

Si hemos elegido como fuente del ataque a TCP se obtendrá una ventana similar a la obtenida para IP e ICMP, solo que con los campos que se tendrán que rellenar para una regla TCP. Estos campos son los campos que se pedían para Wsfep, más los campos requeridos para IP, más los siguientes campos:

- **Puerto Origen:** número de puerto origen del paquete TCP.
- **Puerto Destino:** número de puerto destino del datagrama.
- **Secuencia:** número de secuencia del datagrama TCP.
- **Ack:** valor del campo ACK del paquete TCP.
- **Flags TCP:** es una cadena de unos y ceros que representa a los flags TCP. Los flags son los siguientes: cwr.ece.urg.ack.psh.rst.syn.fin. Deben aparecer escritos en ese orden y sin ningún espacio entre ellos.
- **Ventana:** tamaño de la ventana de la sesión TCP.

Para ignorar cualquiera de los campos descritos para una regla TCP se deben escribir con ceros. Para el caso de los flags TCP se debe escribir una cadena de ocho ceros: “00000000”.

Si la fuente de datos seleccionada es TCP:

Si hemos elegido como fuente del ataque a UDP se mostrará una ventana similar a la obtenida para IP e ICMP, solo que con los campos que se tendrán que rellenar para una regla UDP. Estos campos son los campos comunes para cualquier evento (campos de Wsfep), los campos comunes para cualquier evento de red (los campos de IP) y los campos propios de un datagrama UDP que se muestran a continuación:

- **Puerto Origen:** número de puerto origen del paquete UDP.
- **Puerto Destino:** número de puerto destino del datagrama.
- **Longitud UDP:** tamaño del paquete UDP.

Si se quiere obviar cualquiera de los campos se rellenarán con ceros

AÑADIR UN ATAQUE POTENCIAL:

Cuando se elige añadir un ataque potencial se muestra la siguiente ventana:

Figura 28: Ventana Añadir Ataque potencial.

Para añadir un ataque potencial se requieren los siguientes datos:

- **Identificador del ataque:** número de identificación del ataque, coincide con el identificador de evento. Este campo es obligatorio.

- **Descripción del ataque:** frase que describe al evento.

- **Fuente del ataque:** indica el origen del ataque potencial: ip, icmp o tcp. Al elegir una de las fuentes de ataque, aparece un panel que pide más campos a rellenar. Este panel es el mismo que aparecía para las fuentes de red de la ventana “Añadir Evento” para una regla. Por tanto, los campos y su formato será el mismo que se tenía para ese caso.

Los botones que nos ofrece esta ventana son los siguientes:

- **aceptar:** al hacer clic sobre este botón se escribe el nuevo patrón de ataque en el registro de ataques potenciales.

- **Atrás:** vuelve a la ventana de configuración.

- **Cancelar:** vuelve a la ventana inicial.

5.3 Visualización de los resultados

Si en la ventana inicial del monitor se ha elegido la opción “Últimos Resultados” aparecerá una ventana como esta.

Figura 29: Ventana Últimos Resultados

Para ver los últimos resultados registrados para una sesión determinada, se deberá elegir el nombre del fichero de log adecuado en la lista desplegable. Los ficheros de log aparecen ordenados por fechas.

Tras haber elegido el fichero de log, se muestra el porcentaje de eventos de aplicación y de red registrados. Además se muestra en modo texto la fecha en la que se produjo cada ataque logueado y la descripción del mismo.

5.4 Implementación de la interfaz gráfica

Una vez presentada la parte externa del paquete, tal y como se avanzó en el capítulo 4, en esta sección mostraremos el paquete GUI que contiene todas aquellas clases que son necesarias para crear la interfaz gráfica de usuario.

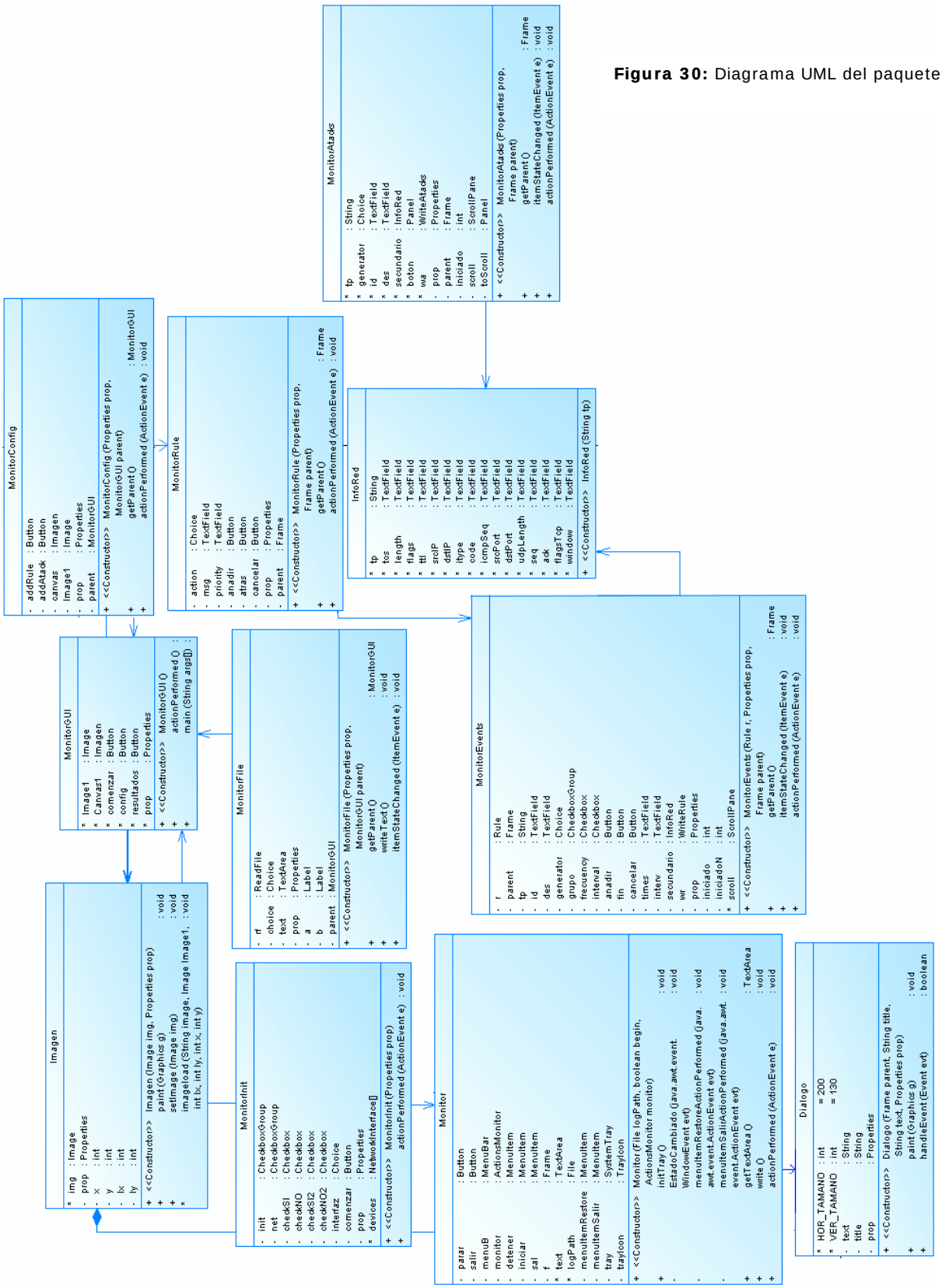


Figura 30: Diagrama UML del paquete GUI

El paquete contiene las siguientes clases:

- **Clase MonitorGUI**

Clase que lanza la aplicación. Muestra una ventana en la que se muestran tres opciones de utilización de la aplicación: configuración del monitor, iniciar sesión y mostrar últimos resultados.

- **Clase MonitorConfig**

Es la clase que se lanza cuando un usuario decide elegir la opción configuración del monitor.

Esta clase proporcionará las herramientas necesarias para que un usuario pueda configurar al monitor de acciones a su gusto: añadir una regla o añadir un nuevo ataque potencial.

- **MonitorRule**

Clase que permite al usuario escribir los datos de una nueva regla a añadir al fichero XML de registro de reglas.

- **MonitorEvents**

Clase invocada desde MonitorRule que permite al usuario escribir los umbrales que no deben superar los distintos eventos que se añaden para una regla determinada.

- **MonitorAtacks**

Clase que permite al usuario añadir los campos necesarios para definir un nuevo ataque potencial que se añadirá al registro XML de ataques potenciales.

- **InfoRed**

Clase que ofrece los campos de texto necesarios para que el usuario escriba los campos de un paquete TCP, UDP o ICMP que será parte de un evento de un ataque potencial o un evento que forme parte de una regla.

- **MonitorInit**

Clase que arranca el monitor de acciones. Permite al usuario decidir si inicia una nueva sesión o no. También permite decidir si se incluye la monitorización del interfaz de red y, si es el caso, elegir el interfaz que se quiere monitorizar.

- **Monitor**

Monitor de acciones. Ventana que muestra todos los resultados que se van obteniendo tras el análisis de los eventos.

- **MonitorFile**

Clase que permite ver los últimos resultados del monitor de acciones. El usuario elegirá una sesión a partir de la fecha de la misma y se mostrarán sus resultados.

- **Imagen**

  Clase que representa una Imagen

- **Dialogo**

  Clase que muestra un diálogo informativo al usuario, cuando le quedan datos por introducir si está configurando el monitor o cuando estos datos no están dentro de los rangos definidos.

  También se mostrarán diálogos cuando se haya producido un ataque y en la acción de la regla del evento aparezca la palabra “msg”.

Capítulo

6

6 Pruebas

En este capítulo se mostrarán las pruebas realizadas para el monitor de acciones.

La funcionalidad del monitor de acciones puede dividirse en cuatro grandes módulos: monitor de eventos de aplicación, monitor de eventos de red, GUI y generales.

- **Monitor de eventos de aplicación:** es aquel módulo que se encarga de monitorizar exclusivamente los eventos que tienen como fuente de datos las aplicaciones Java ME instaladas en el dispositivo.

- **Monitor de eventos de red:** es aquel módulo que tiene como función monitorizar los eventos que llegan a través del interfaz de red.

- **GUI:** interfaz gráfica de usuario gracias a la cual se puede manejar y configurar el monitor de acciones.

- **Generales**: son pruebas destinadas a garantizar el buen funcionamiento global del monitor.

Teniendo en cuenta la clasificación anterior podremos definir cuatro bloques de pruebas: tres bloques correspondientes a los módulos en los que se divide la aplicación más un bloque de prueba de la aplicación global. Así tendremos: pruebas del monitor de aplicación, pruebas del monitor de red, pruebas de GUI y pruebas de la aplicación global o pruebas del monitor de acciones.

Los distintos bloques de pruebas ayudarán a definir las pruebas para cada módulo del monitor de acciones haciendo que sea más sencilla la comprobación de la funcionalidad global de la aplicación desarrollada.

6.1 Pruebas de monitor de aplicación

En esta fase se comprueba que el monitor de eventos de aplicación tenga un funcionamiento tal y como se espera de él. Se consigue probar este módulo tanto en un PC como en una PDA.

Para comprobar que el monitor de aplicación realiza la funcionalidad que se le especifica se utiliza como aplicación de prueba **Wsfep.** Esta aplicación es un software Java ME que comparte ficheros entre dos o más dispositivos limitados que se encuentran el la misma red permitiendo la descarga de los ficheros que el dispositivo tiene marcados como descargables.

Monitorizar esta aplicación puede llegar a ser muy interesante porque permite descargar ficheros que se localizan en otros dispositivos y esto hace que se generen varios eventos críticos, ya que pueden existir usuarios con distintos privilegios de descargas o puede existir un número máximo de descargas por usuario. Cuando alguien tiene esta aplicación instalada en una PDA, le interesará conocer cuándo un usuario ha descargado más ficheros de los permitidos o cuándo un usuario sin privilegios ha descargado un fichero al cual no tenía acceso, por ejemplo. El monitor de eventos de aplicación se encargará de detectar la llegada de todos estos eventos y alertar al cliente.

Wsfep genera un fichero de log en el que se registran todos los eventos críticos que transcurren durante su ejecución. El formato de este fichero de log es un formato válido para el monitor de acciones. Por ello, se ejecutó la aplicación varias veces produciendo varios tipos de eventos para así generar el fichero de log que el monitor de acciones analizará. Por lo tanto, este fichero generado será la entrada de datos del monitor de acciones y formará parte del sensor de eventos de aplicación del mismo.

Tras añadir la entrada de datos al monitor de acciones se prepara una fase de pruebas para el monitor de eventos de aplicación.

En la siguiente tabla se muestra la batería de pruebas aplicada a este módulo y su correspondiente resultado:

Prueba	**Resultado**	**Notas**
Llega un nuevo evento al monitor que no tiene reglas asociadas.	Se loguea el evento en el registro de eventos de aplicación. Si el fichero de registro no está creado da error.	El error se soluciona comprobando si el fichero está creado o no. Si el fichero no está creado, se crea uno nuevo.
Llega un nuevo evento al monitor que tiene una regla asociada.	Si la regla se cumple se toma la acción de la regla. Si la regla no se cumple se registra el evento.	Mismos problemas que en la prueba anterior para el registro y misma solución.
Llega un nuevo evento al monitor que tiene varias reglas asociadas.	Se aplica la regla con mayor prioridad que case con el evento. Si el evento no casa con ninguna regla, se registra.	Mismos problemas que en la prueba anterior para el registro y misma solución.

Prueba	**Resultado**	**Notas**
Llega un evento al monitor que tiene una regla combinada asociada.	Si el/los evento/s con los que se combina se encuentra/n en el registro y todos los eventos casan con la regla, se aplica la acción. Si el/los evento/s no está/n en el registro, se loguea el evento producido. Si el/los evento/s están en el registro pero no casa/n con la regla, se registra el evento.	Problemas de registro al igual que en casos anteriores.
Llega un evento que cumple con regla y está almacenado en el registro.	Se aplica la regla y se elimina evento del registro.	
Llega un evento que cumple con regla y no está almacenado en el registro.	Se aplica la regla.	

6.2 Pruebas de monitor de red

Con estas pruebas queremos saber si se realiza una correcta monitorización de todos los eventos que tienen como origen o salida al exterior el interfaz de red.

Este bloque sólo se puede probar en un PC ya que no se ha encontrado ninguna librería Java para poder detectar tráfico de red en redes inalámbricas y por tanto, las pruebas en una PDA resultan imposibles

En la siguiente tabla se muestran todas las pruebas realizadas y sus resultados. Como se puede observar muchas son comunes a las pruebas del monitor de aplicación:

Prueba	**Resultado**	**Notas**
Llega un nuevo evento al monitor (se detecta un paquete de red).	Si el evento está registrado en el fichero de ataques potenciales se crea un nuevo ataque potencial Si el evento no está en ataques potenciales se ignora.	
Llega un nuevo ataque potencial al monitor que no tiene reglas asociadas.	Se loguea el evento en el registro de eventos de red. Si el fichero de registro no está creado da error.	El error se soluciona comprobando si el fichero está creado o no. Si el fichero no está creado, se crea uno nuevo.
Llega un nuevo ataque potencial al monitor que tiene una regla asociada.	Si la regla se cumple se toma la acción de la regla. Si la regla no se cumple se registra el evento.	Mismos problemas que en la prueba anterior para el registro y misma solución.
Llega un nuevo ataque potencial al monitor que tiene varias reglas asociadas.	Se aplica la regla con mayor prioridad que case con el evento. Si el evento no tiene reglas, se registra.	Mismos problemas que en la prueba anterior para el registro y misma solución.
Llega un ataque potencial al monitor que tiene una regla combinada asociada.	Si el/los evento/s con los que se combina se encuentra/n en el registro y todos los eventos casan con la regla, se aplica la acción. Si el/los evento/s no está/n en el registro, se loguea el evento producido. Si el/los evento/s están en el registro pero no casan con la regla, se loguea el evento.	Problemas de registro al igual que en casos anteriores.
Llega un ataque potencial que cumple con regla y está almacenado en el registro.	Se aplica la regla y se elimina evento del registro.	
Llega un ataque potencial que cumple con regla y no está almacenado en el registro.	Se aplica la regla y no se elimina evento del registro.	
El interfaz de red no está operativo.	No se detecta ningún evento de red.	

6.3 Pruebas del GUI

Con estas pruebas queremos saber si la interfaz de usuario funciona adecuadamente.

Este bloque se consigue probar tanto sobre PC y sobre PDA

En la siguiente tabla se muestran todas las pruebas realizadas:

Prueba	**Resultado**	**Notas**
Se arranca el monitor de acciones.	Aparece la ventana principal del monitor.	La primera versión era demasiado ancha para la PDA.
En la ventana "**Monitor**" se pincha botón "Comenzar".	Aparece la ventana "Comenzar Monitor" con las opciones de arranque de la aplicación.	
En la ventana "**Monitor**" se hace clic sobre el botón "Configuración".	Se muestra la ventana "Configurar" para añadir reglas y ataques potenciales.	La primera versión de "Configurar" era demasiado ancha para la PDA.
En la ventana "**Monitor**" se elige la opción "Últimos Resultados".	Se muestra la ventana "Últimos resultados".	La primera versión de "Últimos resultados" era demasiado ancha para la PDA.
Se cierra la ventana "**Monitor**".	Se sale de la aplicación.	
En la pantalla "**Comenzar Monitor**" se decide: Iniciar Nueva Sesión: SÍ Incluir Sniffer de Red: SÍ Se hace clic en "comenzar".	El monitor de acciones se inicia. Se crea un nuevo fichero de log vacío. Se eliminan los eventos del fichero de registro de red. Se eliminan los eventos del fichero de registro de aplicación. Se monitorizan todos los eventos de aplicación y de red.	En la primera versión existieron problemas con los ficheros de registro ya que no se creaban nuevos. En la primera versión había un único fichero de log para todas las sesiones. Mientras el interfaz de red no esté activo, no se monitoriza ningún evento.
En la pantalla "**Comenzar Monitor**" se decide: Iniciar Nueva Sesión: SÍ Incluir Sniffer de Red: NO Se hace clic en "comenzar".	El monitor de acciones se inicia. Se crea un nuevo fichero de log vacío. Se eliminan los eventos de registro de aplicación. Se eliminan los eventos del fichero de registro de red. Se monitorizan sólo los eventos de aplicación, los de red se ignoran.	Aunque no se inicie el monitor de red, se elimina el contenido del registro de red para ahorrar espacio en disco.
En la pantalla "**Comenzar Monitor**" se decide: Iniciar Nueva Sesión: NO Incluir Sniffer de Red: SÍ Se hace clic en "comenzar".	El monitor de acciones se inicia. Se monitorizan los eventos de aplicación y los de red.	Los ataques se loguean en el fichero de log más antiguo. Los eventos se escriben en el registro a continuación de los eventos registrados en la anterior sesión. Mientras el interfaz de red no esté activo, no se monitoriza ningún evento.

Prueba	**Resultado**	**Notas**
En la pantalla "**Comenzar Monitor**" se decide: Iniciar Nueva Sesión: NO Incluir Sniffer de Red: NO Se hace clic en comenzar.	El monitor de acciones se inicia. Se monitorizan sólo los eventos de aplicación, los de red se ignoran.	Los ataques se loguean en el fichero de log más antiguo. Los eventos se escriben en el registro a continuación de los eventos registrados en la anterior sesión.
En la pantalla "**Comenzar Monitor**" se decide: Iniciar Nueva Sesión: ningún valor Incluir Sniffer de Red: cualquier valor Se hace clic en "comenzar".	La aplicación nos pide que indiquemos si iniciamos sesión.	No hay ningún valor por defecto.
En la pantalla "**Comenzar Monitor**" se decide: Iniciar Nueva Sesión: cualquier valor Incluir Sniffer de Red: ningún valor Se hace clic en "comenzar".	La aplicación nos pide que indiquemos si queremos incluir sniffer de red.	No hay ningún valor por defecto.
En la pantalla "**Comenzar Monitor**" se decide: Iniciar Nueva Sesión: ningún valor Incluir Sniffer de Red: ningún valor Se hace clic en "comenzar".	La aplicación nos pide que indiquemos si iniciamos sesión La aplicación nos pide que indiquemos si queremos incluir sniffer de red.	No hay ningún valor por defecto.
Si estamos en la ventana "**Monitor de Acciones**" y llega un ataque de red o de aplicación.	Aparece un cuadro de diálogo advirtiendo al usuario.	
En la ventana "**Monitor de Acciones**" se pincha en el botón "detener" o en Menú--> detener .	La monitorización de la aplicación y del interfaz de red se detiene. El texto del botón "detener" se cambia por "iniciar". En el menú se desactiva la opción "detener" y se activa "iniciar".	
En la ventana "**Monitor de Acciones**" se pincha en el botón "iniciar" o en Menú--> iniciar .	La monitorización de la aplicación y del interfaz de red y se reinician. El texto del botón "iniciar" se cambia por "detener". En el menú se desactiva la opción "iniciar" y se activa "detener".	
En la ventana "**Monitor de Acciones**" se pincha en el botón "salir" o en Menú--> salir .	Se sale de la aplicación.	
En la ventana "**Monitor de Acciones**" se selecciona en el Menú: Ayuda --> Acerca de...	Se muestra una ventana con ayuda al usuario.	Se muestra una ayuda muy simplificada al usuario.
Se minimiza la ventana "**Monitor de Acciones**".	Se ejecuta la aplicación en *background.*	Funcionalidad añadida para la última versión.
En la ventana "**Configurar**" se hace clic en el botón "Añadir Regla".	Se abre la ventana "Añadir Regla".	

Prueba	**Resultado**	**Notas**
En la ventana "**Configurar**" se hace clic en el botón "Añadir Ataque Potencial de Red".	Se abre la ventana "Añadir Ataque Potencial".	
Se cierra la ventana "**Configurar**".	Se devuelve el control a la ventana "Monitor".	
En la ventana "**Añadir Regla**" se selecciona "Añadir Evento" con un número no entero en "prioridad".	Se abre un diálogo informando del error.	La primera versión no notificaba del error.
En la ventana "**Añadir Regla**" se selecciona "Añadir Evento" con un número entero en "prioridad".	Se leen los campos rellenados por el usuario y se crea una regla.	
En la ventana "**Añadir Regla**" se hace clic en "atrás".	Se devuelve el control a la ventana "Configurar".	Funcionalidad añadida para la última versión.
En la ventana "**Añadir Regla**" se pulsa el botón en "cancelar".	Se devuelve el mando a la ventana "Monitor".	Funcionalidad añadida para la última versión.
Se cierra la ventana "**Añadir Regla**".	Se devuelve el control a la ventana "Monitor".	
En la ventana "**Añadir Evento**" se selecciona una fuente de de evento.	Se añade un panel con un formulario para que el usuario escriba los datos específicos para esa fuente de evento.	Las primeras versiones no aceptaban cambios en la elección de la fuente del evento. Esta ventana es demasiado grande y se soluciona colocando *Scroll Bars.*
En la ventana "**Añadir Evento**" se pulsa el botón "Añadir otro evento" o "finalizar" sin rellenar todos los datos.	Se muestra una ventana de diálogo pidiendo al usuario que introduzca todos los datos.	
En la ventana "**Añadir Evento**" se pulsa el botón "Añadir otro evento" con todos los datos rellenados.	Se añade el evento a la regla creada y se abre otra ventana igual que esta para introducir otro evento a la regla.	
En la ventana "**Añadir Evento**" se pulsa el botón "finalizar" con todos los datos rellenados.	Se añade el evento a la regla creada y se escribe la regla en el fichero XML de reglas. Se muestra de nuevo la ventana "Monitor".	
En la ventana "**Añadir Evento**" se hace clic "cancelar".	Se devuelve el control a la ventana "Añadir Regla".	
Se cierra la ventana "**Añadir Evento**".	Se devuelve el control a la ventana "Monitor".	

Prueba	**Resultado**	**Notas**
En la ventana "**Añadir Ataque Potencial**" se selecciona una fuente del ataque.	Se añade un panel con un formulario para que el usuario escriba los datos específicos para esa fuente de evento.	Las primeras versiones no aceptaban cambios en la elección de la fuente del evento. Esta ventana es demasiado grande y se soluciona colocando *Scroll Bars.*
En la ventana "**Añadir Ataque Potencial**" se pulsa el botón "aceptar" sin rellenar todos los datos.	Se muestra una ventana de diálogo pidiendo al usuario que introduzca todos los datos.	
En la ventana "**Añadir Ataque Potencial**" se pulsa el botón "aceptar" con todos los datos completos.	Se escribe el nuevo ataque potencial en el fichero XML de ataques potenciales.	
En la ventana "**Añadir Ataque Potencial**" se pulsa el botón "atrás".	Se devuelve el control a "Configurar".	
En la ventana "**Añadir Ataque Potencial**" se pulsa el botón "cancelar".	Se devuelve el control a "Monitor".	
Se cierra la ventana "**Añadir Ataque Potencial**".	Se devuelve el control a la ventana "Monitor".	
En la ventana "**Últimos Resultados**" se selecciona un fichero de log cualquiera.	Se devuelve el control a la ventana "Configurar".	
Se cierra la ventana "**Últimos Resultados**".	Se devuelve el control a la ventana "Monitor".	

6.4 Pruebas de monitor de acciones

Tras lanzar el monitor de acciones desde la interfaz gráfica de usuario, se prueba el funcionamiento global de la aplicación. Se prueba solamente en PC, ya que, como vimos anteriormente, no podemos probar la parte de monitor de red.

Para ello, se realizan las siguientes pruebas:

Prueba	**Resultado**	**Notas**
Se inicia el monitor de acciones.	Se empiezan a analizar los eventos de aplicación y los eventos de red.	
Se detiene el monitor de acciones.	Los eventos se siguen produciendo pero el monitor los ignora.	
Se genera un evento de aplicación con una regla asociada.	Si la regla aplica, se ejecuta su acción. Si no aplica, se registra el evento en el registro de eventos de aplicación.	
Se genera un evento de aplicación sin regla asociada.	Se guarda evento en el registro de eventos de aplicación.	Se guarda por si el evento se combina con otro evento de aplicación o de red.
Se genera un evento de aplicación que se combina con otros eventos de aplicación.	Se busca el estado de todos los eventos, si todos cumplen la regla se aplica la regla y se elimina evento producido del registro.	
Se genera un evento de aplicación que se combina con otros eventos de red.	Se busca el estado de todos los eventos de red en el registro de eventos de red, si es que los eventos están registrados. Si todos los eventos cumplen la regla, sea aplica y se elimina del registro de eventos de aplicación el evento producido.	
Se genera un evento de aplicación que se combina con otros eventos de red y de aplicación.	Se busca el estado de todos los eventos de red en el registro de eventos de red y el estado de los eventos de aplicación en el registro de eventos de aplicación, si es que los eventos están registrados. Si todos los eventos cumplen la regla, se aplica y se elimina el evento producido del registro de eventos de aplicación.	

Prueba	Resultado	Notas
Llega un evento de red al monitor.	Se comprueba si es ataque potencial. Si es ataque potencial se analiza el evento, si no, se ignora.	
Llega un ataque potencial de red que tiene regla asociada.	Si la regla aplica, se ejecuta su acción. Si no aplica, se registra el evento en el registro de eventos de red dentro del elemento adecuado.	
Se genera un ataque potencial de red sin regla asociada.	Se guarda evento en el registro de eventos de red dentro del elemento adecuado.	Se guarda por si el evento se combina con otro evento de aplicación o de red.
Se genera un ataque potencial de red que se combina con otros ataques potenciales de red.	Se busca el estado de todos los eventos, si todos cumplen la regla se aplica la regla y se elimina evento producido del registro de red.	
Se genera un ataque potencial que se combina con otros eventos de aplicación.	Se busca el estado de todos los eventos de aplicación en el registro de eventos de aplicación, si es que los eventos están registrados. Si todos los eventos cumplen la regla, sea aplica y se elimina del registro de eventos de red el ataque potencial.	
Se genera un evento de red que se combina con otros eventos de aplicación y otros ataques potenciales de red.	Se busca el estado de todos los eventos de aplicación en el registro de eventos de aplicación y el estado de los ataques potenciales de red en el registro de eventos de red, si es que los eventos están registrados. Si todos los eventos cumplen la regla, sea aplica y se elimina el ataque potencial producido del registro de eventos de aplicación.	

6.5 Problemas encontrados

Aplicando las distintas pruebas que mostradas en los apartados anteriores se encontraron los siguientes problemas o dificultades:

Por un lado, no se encuentra ninguna librería Java para detectar el tráfico de red para dispositivos limitados. En la búsqueda de librerías se encuentra JPCAP que permite detectar y analizar el tráfico de red. El problema que presenta esta librería es que no admite cualquier tarjeta de red y tiene grandes dificultades para detectar el tráfico en dispositivos inalámbricos. Por ello, la parte que corresponde con el monitor de red no se puede probar.

Por otro lado, se tienen problemas con la aplicación WSFEP, ya que el funcionamiento en entornos inalámbricos es indeterminado. La aplicación si se conseguía ejecutar en PDAs o PCs pero los dispositivos algunas veces no se veían entre sí o no se podía realizar una de las funciones principales de la aplicación como la de descargar ficheros en remoto.

Se intenta ejecutar WSFEP en distintas configuraciones: una red ad-hoc, una red LAN, una red inalámbrica, una configuración de un servidor más dos host y en todos los casos se obtiene el mismo resultado no estable, lo cual dificultaba la generación de ficheros de logs por parte de la aplicación.

Capítulo

7

7 Historia del Proyecto

En este capítulo se muestra la evolución del proyecto dividiendo éste en fases, según fue creciendo y formándose.

El proyecto fue desarrollado teniendo en cuenta distintas fases y etapas de modo que se pueda ver su crecimiento y evolución y cómo se llega a la solución final que aquí se presenta.

Las fases en las que se ha desarrollado el proyecto se muestran a continuación:

7.1 Fase I: Definición de objetivos y requisitos

En esta primera fase se realiza un análisis de los objetivos que se pretenden conseguir para el proyecto y los requisitos que se deben cumplir para el mismo.

El resumen de objetivos y requisitos definidos se muestran a continuación:

Se pretende desarrollar un monitor de acciones que pueda ser ejecutado en dispositivos limitados y que sea capaz de detectar anomalías que producen determinadas aplicaciones software instaladas en el dispositivo y anomalías que pueden llegar a través del interfaz de red.

El monitor de acciones debe poder detener la detección de anomalías para el interfaz de red si el dispositivo limitado no tuviera conexión a Internet.

También debe poder detectar las anomalías que se produzcan por la combinación de acciones que llegan a través de las aplicaciones software o a través del interfaz de red.

El monitor de acciones debe presentar una interfaz gráfica sencilla que permita al usuario final observar todas las anomalías detectadas. La aplicación debe tener una interfaz de uso fácil y auto explicativa de modo que el cliente no tenga problemas con su uso.

Adicionalmente el interfaz gráfico debe permitir al usuario configurar el monitor a su gusto especificando las anomalías que quiere que se detecten. Si el usuario no configura nunca el monitor de acciones, éste debe ser lo más autónomo posible en la detección de anomalías o ataques. Por ello, tendrá almacenadas un conjunto de reglas sobre ataques comunes.

Por último, la aplicación desarrollada debe de cumplir con todas las condiciones que debe cumplir cualquier aplicación desarrollada para un dispositivo limitado: debe consumir pocos recursos de memoria y disco. Debe ser desarrollado en un lenguaje que se pueda ejecutar en el dispositivo y no debe ser demasiado extensa.

7.2 Fase II: Estudio de las herramientas y fuentes de información utilizadas

Para poder desarrollar el monitor de acciones era necesario la búsqueda y estudio de determinadas herramientas que permitieran conseguir los objetivos y requisitos especificados en la primera fase.

Lo primero de todo era realizar un estudio de los monitores de acciones que existen en el mercado actual. Este estudio no sacó grandes conclusiones ya que hay poca información sobre monitores de acciones. Por ello, se enfoca el estudio al campo de los detectores de intrusos, también conocidos como IDS.

Gracias al estudio de los IDS, descubro que los IDS basados en red tienen mucho que ver con la parte de detección de eventos de Internet que se pretende desarrollar ya que comparten objetivos, características y parte de funcionalidad. Por lo tanto, se estudia la arquitectura de los IDS basados en red con objeto de modificarla y adaptarla al monitor de acciones.

Tras el estudio genérico de los IDS, se realiza una investigación sobre un estándar comercial *open source* de detección de intrusos basado en red llamado Snort. Snort es una aplicación instalable en entornos Windows y Linux que monitoriza únicamente los eventos de red. Esta aplicación decodifica todos los paquetes que llegan por el interfaz de red y decide si éstos son ataques o no basándose en un sistema de reglas que tiene definido. Para poder realizar todas estas operaciones se apoya en una pequeña base de datos para almacenar eventos.

Como Snort cubre una pequeña parte del proyecto que aquí se define, se toma como ejemplo su sistema de reglas y base de datos para aplicarlo al monitor de acciones. Pero hay un inconveniente: ambos se apoyan en ficheros o bases de datos demasiado extensas y esto aplicado al monitor de acciones produciría una sobrecarga en el sistema. Con lo cual, se mantiene la idea de tener un motor de reglas y una base de datos pero no con el formato de Snort.

Para solucionar el problema de la definición del formato de las reglas y del almacén de eventos de cualquier tipo se propone el uso de ficheros XML con formato definido exclusivamente para el monitor para así guardar exclusivamente lo estrictamente necesario. Ahora surgía otro tema más: cómo leer y escribir dichos documentos. Esto se solucionó con la búsqueda de un estándar de parseo XML válido para Java y válido para dispositivos limitados. De este modo se concluye en que KXML es la librería más apropiada para este fin.

El siguiente paso a realizar era definir cuál sería el lenguaje de programación utilizado para desarrollar la aplicación que formaría el monitor de acciones. Se decide utilizar Java debido al gran abanico de posibilidades que éste ofrece y debido a que se puede ejecutar en cualquier sistema operativo previa configuración de la máquina virtual. Pero Java tiene varias plataformas y, por tanto, se debía hilar más fino buscando aquella plataforma que permitiera desarrollar una aplicación para dispositivos limitados con interfaz gráfica de usuario. Teniendo en cuenta lo anterior, se elige Java Personal Profile como lenguaje de programación del monitor de acciones.

Por último se realiza un estudio de uno de los temas más importantes: descubrir cuáles son los ataques de Internet más comunes y cuáles de ellos pueden llegar a un dispositivo limitado. Para cada ataque encontrado se estudia su definición y cuáles son los umbrales que al superar formarían un ataque.

7.3 Fase III: Diseño

La siguiente fase del proyecto es el diseño de la aplicación que se pretende desarrollar.

Para diseñar el monitor de acciones se tiene en cuenta todo lo estudiado en la fase anterior.

Se pretende realizar un diseño basado en bloques, partiendo de la arquitectura de un IDS basado en red. La separación en bloques hace que se pueda definir una funcionalidad específica para cada bloque y se pueda aplicar el modelo Java de Orientación a Objetos de modo que cada bloque significará un paquete Java.

Como ya se ha comentado la arquitectura basada en bloques facilita el diseño de un diagrama de clases. Se divide la aplicación en paquetes de modo que cada paquete tiene la funcionalidad que se define para el modelo de bloques.

Tras haber diseñado un diagrama de clases en formato UML se crea un diagrama de flujo en el que se muestran las acciones que se deben realizar y las entradas y salidas que se toman en los distintos pasos que se toman a lo largo de todo el proceso.

7.4 Fase IV: Implementación

La implementación del prototipo de monitor de acciones es el siguiente paso que se debe tomar. Para facilitar el desarrollo se divide la implementación en varias subfases: implementación del monitor de eventos de aplicación, implementación del monitor de eventos de red e implementación del monitor de acciones.

7.4.1 Implementación del monitor de eventos de aplicación

Es la primera etapa que se cubre dentro de la implementación y desarrollo de la aplicación.

En esta subfase se crean todas las clases que definirán un evento de aplicación, el estado del evento, las reglas que se pueden asociar al evento y las clases que sean capaces de leer y escribir en el registro de eventos de aplicación.

Tras haber definido los objetos del monitor de aplicación se utiliza un fichero de log de la aplicación que se monitoriza. Se desarrolla un primer monitor que será el que lea todos los eventos logueados para la aplicación y compruebe si dichos eventos son ataques o no. Si este primer monitor encuentra un ataque, éste es logueado.

7.4.2 Implementación del monitor de eventos de red

Tras haber desarrollado el monitor de eventos de aplicación, se sigue la misma filosofía para implementar el monitor de eventos de red: se crean clases que representen a los distintos tipos de eventos de red según el protocolo, se definen reglas de eventos de aplicación y se implementan las clases que son capaces de leer y escribir en los ficheros XML de registro de eventos y de registro de patrones de ataques de red.

Para los eventos de red no existe ningún fichero de log en el que aparezcan registrados todos los paquetes de red que llegan al dispositivo. Por ello, era necesario también implementar un sniffer de red que fuera capaz de leer y procesar toda la información que saliese o entrase por el interfaz de red.

Una vez desarrollado el sniffer se desarrolla una clase capaz de detectar todos los eventos que fueran ataques potenciales haciendo el uso de los patrones definidos.

Tras haber detectado el ataque potencial era necesario saber si dicho ataque potencial suponía un ataque o no. Para ello, se desarrollan las clases que representan al monitor de eventos de red. Una vez detectado el ataque, se loguea.

7.4.3 Implementación del monitor de acciones

Una vez se han desarrollado los dos monitores por separado, se juntan para comprobar que su funcionalidad conjunta es la deseada y que cumple con los requisitos del monitor de acciones.

Se añade un gestor de reglas para incluir todas aquellas reglas que, para su cumplimiento se necesitan uno o varios eventos de red y uno o varios eventos de aplicación.

Por otro lado, se obliga a que todos los eventos, ya sean de red o de aplicación sean logueados en el mismo fichero de log (el cual es único para cada sesión de monitoreo que se realiza)

7.5 Fase V: Implementación del Interfaz de Usuario

En esta fase se implementa la interfaz grafica de la aplicación, gracias a la cual el usuario podrá utilizar el monitor de acciones: iniciando monitorizaciones,

deteniéndolas, viendo los últimos ficheros de log registrados, configurando reglas y ataques potenciales.

Se desarrollan todas las ventanas que permiten interactuar al usuario dividiéndolas en tres grupos: las ventanas que permiten configurar el monitor de acciones, las ventanas que permiten ver los últimos resultados y las ventanas que nos permiten arrancar el monitor de acciones y establecer nuevas sesiones de monitorización.

7.6 Fase VI: Fase de Pruebas

Tras haber implementado la aplicación era necesario probar su funcionalidad. Para ello se prepara una batería de pruebas: pruebas de interfaz gráfica, pruebas de funcionalidad, pruebas de usuario, etc.

En la fase de pruebas se descubren fallos de implementación que es necesario corregir. Por ello, esta fase se solapó en ciertos momentos con la fase de implementación.

Para poder probar la funcionalidad del monitor de acciones era necesario añadir la aplicación que se pretendía monitorizar. Wsfep es una aplicación utilizada para compartir ficheros entre varios dispositivos con lo cual daba mucho juego para poder monitorizar los eventos que en ella se loguearan.

Tras la instalación de la aplicación se generaron distintos ficheros de log que servirían al monitor de acciones para leerlos y descubrir los posibles ataques que se pudieran esconder en él.

Para probar el monitor de red se generaron a mano distintos eventos de red como: inicio de sesiones FTP, ataques de ping flooding, etc. Estos eventos llegaban al monitor y era éste quien decidía si eran ataques o no.

Las pruebas de usuario realizadas consistieron el probar la robustez de la interfaz gráfica: el uso de botones, menú, paneles, etc. También se probó la claridad con la que se mostraban los datos al usuario.

7.7 Fase VIII: Documentación

La última fase del proyecto, y no por ello la menos importante, fue la fase de documentación.

Esta fase a su vez se dividen dos: documentación de las clases Java implementadas y elaboración de la memoria del proyecto.

Para la escritura de la memoria del proyecto se recopilaron, resumieron y tradujeron toda aquella información que se utilizó para el desarrollo del proyecto: información estudiada e información creada para el proyecto como dibujo de la arquitectura, diagramas de flujo, imágenes de la aplicación, esquemas de los documentos XML, etc.

7.8 Diagrama de Gantt

A continuación se presenta un diagrama de Gantt mostrando las etapas del proyecto en las que se dividió el proyecto durante los **9 meses** en los que se elaboró:

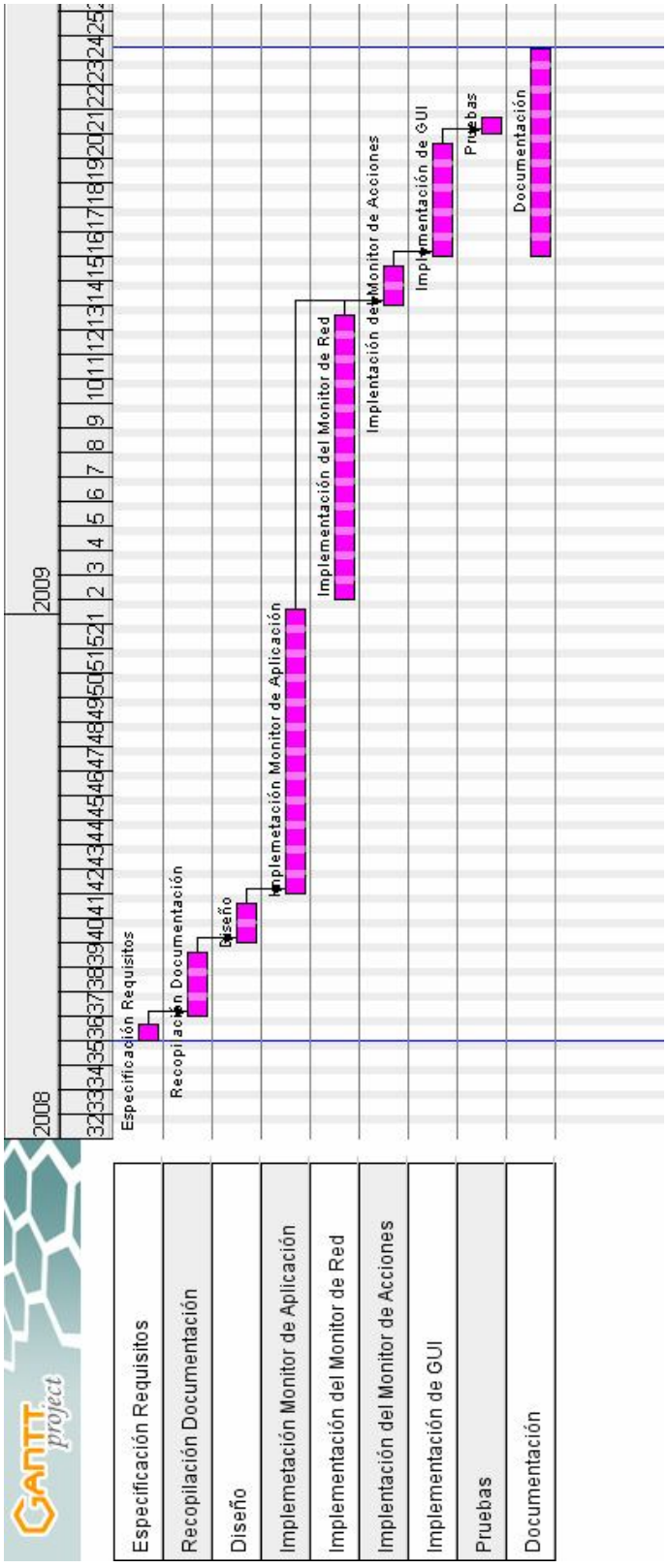


Figura 31: Diagrama de Gantt

En el diagrama anterior vemos que se han definido 9 tareas, las cuales coinciden con las fases definidas para el proyecto.

A continuación se muestra una tabla con el desglose de cada tarea o fase:

Nombre Tarea	Fecha Inicio	Fecha Fin	Días Laborales
Especificación de Requisitos	01/09/2008	06/09/2008	5
Recopilación de Documentación	08/09/2008	27/09/2008	17
Diseño del Proyecto	29/09/2008	11/10/2008	10
Implementación del monitor de aplicación	13/10/2008	03/01/2009	72
Implementación del monitor de red	05/01/2009	28/03/2009	72
Implementación del monitor de acciones	30/03/2009	11/04/2009	12
Implementación de GUI	13/04/2009	16/05/2009	30
Pruebas	18/05/2009	23/05/2009	5
Documentación	13/04/2009	12/06/2009	53

Se han empleado semanas de 6 días laborales

7.9 Presupuesto

En este apartado se muestra el presupuesto del proyecto desglosado en presupuesto material y presupuesto de mano de obra.

PRESUPUESTO MATERIAL:

En la siguiente tabla mostramos el desglose del presupuesto material:

Material	Precio/Unidad €	Unidades	Precio Total €
Ordenador de sobremesa	600	1	600
Conexión mensual a Internet	30	10	300
PDA	175	2	350

Presupuesto Material Total = 1250 €

PRESUPUESTO DE MANO DE OBRA

Para calcular este presupuesto se ha calculado el número de horas empleadas para cada fase. Este número se saca a partir del número de días empleados para cada fase. Para ello se cuentan semanas de 6 días (un día libre a la semana) y 8 horas de trabajo diario

PRESUPUESTO TOTAL DEL MONITOR DE ACCIONES

En la siguiente tabla mostramos el desglose del presupuesto de mano de obra:

Nombre de la tarea	Fecha de Inicio	Fecha de Fin	Días	Precio €
Especificación de Requisitos	01/09/2008	06/09/2008	5	1200
Recopilación de Documentación	08/09/2008	27/09/2008	17	4080
Diseño	29/09/2008	11/10/2008	10	2400
Implementación del monitor de aplicación	13/10/2008	03/01/2009	72	17280
Implementación del monitor de red	05/01/2009	28/03/2009	72	17280
Implementación de monitor de acciones	30/03/2009	11/04/2009	12	2880
Implementación de GUI	13/04/2009	16/05/2009	30	7200
Pruebas	18/05/2009	23/05/2009	5	1200
Documentación	13/04/2009	12/06/2009	17 *	4080

Presupuesto Mano de Obra Total = 57600 €

* Se cuentan los días que no solapan con las fases anteriores.

Sumando las dos las dos cantidades anteriores obtenemos el presupuesto total del proyecto que mostramos a continuación:

Presupuesto Material	1.250 €
Presupuesto de Mano de Obra	57.600 €
Total	58.850 €

Capítulo

8

8 Conclusiones y líneas futuras

8.1 Conclusiones del proyecto

Como se ha mostrado a lo largo de toda esta memoria, el monitor de acciones es una herramienta muy potente para detectar anomalías y ataques procedentes de la red Internet. Cualquier usuario del monitor puede aprovechar las posibilidades que éste ofrece respecto a reglas y patrones y definirse su propio sistema de seguridad. La complejidad de este sistema de seguridad puede ir creciendo según crecen los concomimientos del usuario sobre los protocolos de Internet, ya que de este modo puede elaborar reglas complejas y que cubran importantes ataques de la red.

La cobertura del monitor de acciones se puede combinar con la utilización de un *firewall* y/o un antivirus. De este modo, el usuario cualificado del que hablábamos antes puede crear su propio sistema de seguridad más o menos robusto.

El monitor de acciones se basa en un amplio sistema de reglas. Se dice que es amplio ya que permite definir y combinar varios tipos de reglas. Esto va a permitir que el monitor de acciones tenga el alcance que se desee tan solo con cambiar el valor de las reglas. Es decir, el Sistema de Reglas es lo suficientemente amplio y genérico como para poder llevarlo a otros ámbitos que no sean la monitorización del interfaz de red o la monitorización de determinadas aplicaciones software.

El Sistema de Reglas, podrá ser aplicado a la detección de cualquier tipo de anomalía de cualquier tipo de sistema.

Si seguimos en el ámbito de los dispositivos limitados, podríamos utilizar el sistema de reglas para definir reglas que nos permitieran detectar, por ejemplo, un consumo exhaustivo de determinados recursos como, por ejemplo, la batería del dispositivo.

Por otro lado, también podemos aplicar el Sistema de Reglas a otros ámbitos que no estén relacionados con los dispositivos limitados, como por ejemplo para cualquier aplicación empresarial basada en reglas y o umbrales en el campo de las Telecomunicaciones, Informática o, por ejemplo, Banca. En este último caso el

Sistema de Reglas podría ser utilizado para la concesión de un crédito a un cliente del banco. Si el cliente cumple con las condiciones que se definen en la regla, será premiado con la concesión del crédito, pero si, por otro lado no cumple las condiciones no obtendrá el crédito.

Teniendo en cuenta las explicaciones anteriores podemos concluir en que el monitor de acciones nos puede ayudar a descubrir infinidad de ataques en la red y que su Sistema de Reglas nos puede ayudar a resolver cualquier tipo de problema en el que intervengan umbrales y decisiones.

8.2 Trabajo futuro

Tras haber desarrollado de prototipo de monitor de acciones para dispositivos limitados, se proponen varias acciones futuras para mejorarlo:

Acciones para optimizar el sistema de ficheros:

- El monitor de acciones es una aplicación diseñada para dispositivos limitados. Por lo tanto, es necesario que el consumo de memoria y de espacio en disco sean lo mínimos posibles.

  Desarrollando un formato XML personalizado se ha conseguido obtener un formato de fichero lo más mínimo posible y que además permite la utilización de una librería para su parseo que no consume demasiados recursos de memoria.

  Aún así, esto no es suficiente y se propone la búsqueda de un **formato óptimo** o una **compresión** mínima tanto para los ficheros XML que componen la "mini base de datos", como para los ficheros de log que se generan.

- Otra idea para aprovechar el poco espacio en disco que nos ofrece este tipo de dispositivo es definir una política de borrado de los ficheros de log, de modo que se pueda especificar el tiempo de vida máximo de un fichero de log dentro de la memoria del dispositivo.

Acciones para mejorar el análisis del tráfico de red

- Una propuesta adicional de acción futura a es la búsqueda y uso de un sniffer de red disponible para cualquier tipo interfaz inalámbrico ya que la librería Jpcap utilizada tiene ciertas limitaciones en este aspecto: sólo acepta determinadas tarjetas inalámbricas que funcionen en la plataforma Linux.

Acciones para facilitar la presentación de datos en el monitor:

- En la versión del monitor de acciones que aquí se presenta se muestran los datos a través de un fichero de log y a través de cuadros de diálogo que aparecen en el momento en que llega un ataque. También se muestra el porcentaje de eventos que tienen como fuente la aplicación y el porcentaje de eventos que llegan por el interfaz de red.

- Sería buena idea poder representar todos estos resultados de manera gráfica empleando algún diagrama estadístico, como por ejemplo un diagrama de sectores con los porcentajes de los eventos o un polígono de frecuencias para ver la evolución de los ataques dentro de un intervalo de tiempo determinado.

De esta forma se ayuda al usuario a entender los datos y verlos con un simple golpe de vista. Además la interfaz gráfica sería más llamativa.

- Otra buena práctica sería poder realizar filtros de modo que un usuario pueda ver todos los eventos que se han generado dentro de un intervalo de tiempo determinado o los ataques que ha generado un determinado usuario o una fuente de eventos específica.

Acciones que amplían funcionalidad:

- Se propone que el monitor de acciones pueda interactuar con otras aplicaciones. En esta primera versión de monitor de acciones todas las acciones que se realizan tras la detección de un evento son informativas. Sería buena idea que, dependiendo del ataque producido, se tomara una acción correctiva concreta. Un posible resultado ante un ataque podría ser la notificación a cualquier aplicación o recurso afectado o implicado en dicho ataque.

- También se podría notificar al usuario por medio de un mensaje de correo electrónico o SMS de la llegada de un ataque.

- Las dos acciones anteriores pueden servir para que determinadas aplicaciones, recursos o usuarios puedan tomar medidas en función de la gravedad del ataque y de ese modo, evitar posibles reincidencias del mismo. Además la llegada de un ataque puede servir de patrón de aprendizaje para evitar posibles ataques relacionados con él.

Medidas

- Para poder saber lo eficiente que es la aplicación y la cantidad de recursos que consume del dispositivo, sería buena idea poder tomar medidas que reflejen la respuesta del monitor de acciones: tiempo que tarda en determinar si un evento es un ataque o no, tiempo que tarda en lanzar una alerta tras detectar un ataque, cantidad de memoria consumida, porcentaje de tamaño que ocupan los ficheros de registro almacenados en disco, etc.

Capítulo

9

9 Bibliografía

Referencias:

[1] Mobile Network Security
Y. Xiao, X. Shen and D.-Z Du, 2006. Wireless/Mobile Network Security. Editorial Springer

[2] Govannom
http://www.govannom.org/seguridad/protect/ids_idp/arkitectura_ids.pdf

[3] Descripción general de los sistemas de detección de intrusos
http://catarina.udlap.mx/u_dl_a/tales/documentos/lem/urbina_p_j/capitulo2.pdf

[4] Sistema de detección de intrusos – Wikipedia
http://es.wikipedia.org/wiki/Sistema_de_detecci%C3%B3n_de_intrusos

[5] Common intrusion detection framework
http://gost.isi.edu/cidf/

[6] Intrusion detection exchange format (idwg)
http://www.ietf.org/html.charters/OLD/idwg-charter.html

[7] Snort
http://www.snort.org

[8] Actions monitor
http://www.entrebits.com/software/actions-monitor

[9] Copianos.com
http://www.copianos.com/es/articulos/tecnologia/seguridad-en-dispositivos-moviles

[10] Battery-Based Intrusion Detection
http://scholar.lib.vt.edu/theses/available/etd-04212005-120840/unrestricted/JacobyBatteryBasedIntrusionDetection.pdf

[11] Ataque Smurf – Wikipedia
http://es.wikipedia.org/wiki/Smurf_ataque

[12] Apuntes de la asignatura "Software de Comunicaciones" de Ingeniería Técnica en Telecomunicación especialidad, Telemática.

[13] Java a tope: J2ME
http://www.lcc.uma.es/~galvez/J2ME.html

[14] Sun J2ME personal profile
http://java.sun.com/products/personalprofile/overview.html

[15] KXML
http://kxml.sourceforge.net/

[16] JPCAP
http://netresearch.ics.uci.edu/kfujii/jpcap/doc/

[17] WSFEP
https://bach.gast.it.uc3m.es/~gditeniz/

[18] Tecnoplan- todo para PDAs
http://www.tecnopalm.com/modules.php?name=News&file=article&sid=1882

[19] About.com
http://mobileoffice.about.com/gi/dynamic/offsite.htm?zi=1/XJ&sdn=mobileoffice&cdn=gadgets&tm

[20] About.com - avast!
http://mobileoffice.about.com/gi/dynamic/offsite.htm?zi=1/XJ&sdn=mobileoffice&cdn=gadgets&tm=80&gps=82_5_1003_569&f=00&su=p284.9.336.ip_p504.1.336.ip_&tt=3&bt=0&bts=0&zu=http%3A//www.avast.com/eng/avast_4_pda.html

[21] emule-protocol
https://bach.gast.it.uc3m.es/~gditeniz/emule-protocol.pdf

[22] Pervasive discovery protocol (PDP)
https://bach.gast.it.uc3m.es/~gditeniz/draft_pdp.pdf

[23] SAX
http://www.saxproject.org/

[24] Actions Monitor 1.02
http://actions-monitor.programas-gratis.net/

APÉNDICE A: SCHEMAS DEFINIDOS

Schema de ataques potenciales: PotencialAtacks.xsd

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
	<xsd:element name="ip">
		<xsd:complexType>
			<xsd:sequence>
				<xsd:element name="ipAtack">
					<xsd:complexType>
						<xsd:sequence>
							<xsd:element name="tos"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
							<xsd:element name="length"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
							<xsd:element name="flags"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
							<xsd:element name="ttl"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
							<xsd:element name="srcIP"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
							<xsd:element name="dstIP"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
						</xsd:sequence>
						<xsd:attribute name="id"
type="xsd:string"/>
						<xsd:attribute name="description"
type="xsd:string"/>
					</xsd:complexType>
			</xsd:sequence>
		</xsd:complexType>
	</xsd:element>
	<xsd:element name="icmp">
		<xsd:complexType>
			<xsd:sequence>
				<xsd:element name="icmpAtack">
					<xsd:complexType>
						<xsd:sequence>
							<xsd:element name="tos"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
							<xsd:element name="length"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
							<xsd:element name="flags"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
							<xsd:element name="ttl"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
							<xsd:element name="srcIP"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
							<xsd:element name="dstIP"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
							<xsd:element name="type"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
							<xsd:element name="code"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
							<xsd:element name="icmpSeq"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
						</xsd:sequence>
						<xsd:attribute name="id"
type="xsd:string"/>
						<xsd:attribute name="description"
type="xsd:string"/>
```

```
                                </xsd:complexType>
                            </xsd:element>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
                <xsd:element name="tcp">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="tcpAtack">
                                <xsd:complexType>
                                    </xsd:sequence>
                                        <xsd:element name="tos"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
                                        <xsd:element name="length"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
                                        <xsd:element name="flags"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
                                        <xsd:element name="ttl"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
                                        <xsd:element name="srcIP"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
                                        <xsd:element name="dstIP"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
                                        <xsd:element name="srcPort"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
                                        <xsd:element name="dstPort"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
                                        <xsd:element name="seq"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
                                        <xsd:element name="ack"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
                                        <xsd:element name="tcpFlags"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
                                        <xsd:element name="window"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
                                    </xsd:sequence>
                                    <xsd:attribute name="id"
type="xsd:string"/>
                                    <xsd:attribute name="description"
type="xsd:string"/>
                                </xsd:complexType>
                            </xsd:element>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
                <xsd:element name="udp">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="udpAtack">
                                <xsd:complexType>
                                    <xsd:sequence>
                                        <xsd:element name="tos"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
                                        <xsd:element name="length"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
                                        <xsd:element name="flags"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
                                        <xsd:element name="ttl"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
                                        <xsd:element name="srcIP"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
                                        <xsd:element name="dstIP"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
                                        <xsd:element name="srcPort"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
                                        <xsd:element name="dstPort"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
                                        <xsd:element name="udpLength"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
                                    </xsd:sequence>
```

```
                                        <xsd:attribute name="id"
type="xsd:string"/>
                                        <xsd:attribute name="description"
type="xsd:string"/>
                                </xsd:complexType>
                        </xsd:element>
                </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="PotencialAtacks">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="ip" minOccurs="0" maxOccurs="1" />
                <xsd:element ref="icmp" minOccurs="0" maxOccurs="1"
/>
                <xsd:element ref="tcp" minOccurs="0" maxOccurs="1"
/>
                <xsd:element ref="udp" minOccurs="0" maxOccurs="1"
/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

Schema de Registro de Eventos de Aplicación: AplRegister.xsd

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="lastDay" type="xsd:date"/>
<xsd:element name="lastHour" type="xsd:string"/>
<xsd:element name="firstDay" type="xsd:date"/>
<xsd:element name="firstHour" type="xsd:string"/>
<xsd:element name="times"  type="xsd:integer"/>
<xsd:element name="interval" type="xsd:decimal"/>

<xsd:element name="event">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="lastDay" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="lastHour" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="firstDay" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="firstHour" minOccurs="1" maxOccurs="1"/>
        </xsd:sequence>
        <xsd:choice>
            <xsd:element name="minInterval">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element ref="times" minOccurs="1"
maxOccurs="1"/>
                        <xsd:element ref="interval"
minOccurs="1" maxOccurs="1"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="frecuency">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element ref="times" minOccurs="1"
maxOccurs="1"/>
                        <xsd:element ref="interval"
minOccurs="1" maxOccurs="1"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:choice>
```

```
            <xsd:attribute name="id" use="required"/>
            <xsd:attribute name="generator" use="required"/>
            <xsd:attribute name="description" use="required"/>
      </xsd:complexType>
 </xsd:element>

 <xsd:complexType name="tuser">
    <xsd:sequence>
      <xsd:element ref="event" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:ID" use="required"/>
  </xsd:complexType>

  <xsd:element name "taplication1">
      <xsd:sequence>
      <xsd:element name="user" type="tuser" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:element>

<xsd:element name="AplRegister" >
      <xsd:complexType>
            <xsd:sequence>
                  <xsd:element name="aplication1" type="taplication1"
minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
      </xsd:complexType>
</xsd:element>

</xsd:schema>
```

Schema de Registro de Eventos de Red: NetworkRegister.xsd

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="times"  type="xsd:integer"/>
<xsd:element name="interval" type="xsd:decimal"/>

<xsd:element name="minInterval">
      <xsd:complexType>
            <xsd:sequence>
                  <xsd:element ref="times" minOccurs="1" maxOccurs="1"/>
                  <xsd:element ref="interval" minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
      </xsd:complexType>
</xsd:element>
<xsd:element name="frecuency">
      <xsd:complexType>
            <xsd:sequence>
                  <xsd:element ref="times" minOccurs="1" maxOccurs="1"/>
                  <xsd:element ref="interval" minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
      </xsd:complexType>
</xsd:element>

      <xsd:element name="networkRegister">
            <xsd:complexType>
                  <xsd:sequence>
                        <xsd:element name="ip">
                              <xsd:complexType>
                                    <xsd:sequence>
                                          <xsd:element name="user">
                                                <xsd:complexType>
                                                      <xsd:sequence>
```

```
                                                                        <xsd:element
name="event">
       <xsd:complexType>
       <xsd:sequence>
       <xsd:element name="lastDay" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
       <xsd:element name="lastHour" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
       <xsd:element name="firstDay" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
       <xsd:element name="firstHour" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
       <xsd:element name="description" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
       <xsd:element ref="frecuency"/>
       <xsd:element ref="minInterval"/>
       <xsd:element name="tos" type="xsd:string" minOccurs="1" maxOccurs="1"/>
       <xsd:element name="length" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
       <xsd:element name="flags" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
       <xsd:element name="ttl" type="xsd:string" minOccurs="1" maxOccurs="1"/>
       <xsd:element name="srcIP" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
       <xsd:element name="dstIP" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
       </xsd:sequence>
       <xsd:attribute name="id" type="xsd:string" use="required"/>
       <xsd:attribute name="generator" type="xsd:string" use="required"/>
       <xsd:attribute name="description" type="xsd:string" use="required"/>
       </xsd:complexType>
       </xsd:element>
                                                              </xsd:sequence>
                                                              <xsd:attribute
name="id" type="xsd:string" use="required"/>
                                                      </xsd:complexType>
                                               </xsd:element>
                                        </xsd:sequence>
                                 </xsd:complexType>
                          </xsd:element>
                          <xsd:element name="icmp">
                                 <xsd:complexType>
                                        <xsd:sequence>
                                               <xsd:element name="user">
                                                      <xsd:complexType>
                                                             <xsd:sequence>
                                                                    <xsd:element
name="event">
```

```
      <xsd:complexType>

      <xsd:sequence>

      <xsd:element name="lastDay" type="xsd:string" minOccurs="1"
maxOccurs="1"/>

      <xsd:element name="lastHour" type="xsd:string" minOccurs="1"
maxOccurs="1"/>

      <xsd:element name="firstDay" type="xsd:string" minOccurs="1"
maxOccurs="1"/>

      <xsd:element name="firstHour" type="xsd:string" minOccurs="1"
maxOccurs="1"/>

      <xsd:element ref="frecuency"/>

      <xsd:element ref="minInterval"/>

      <xsd:element name="tos" type="xsd:string" minOccurs="1" maxOccurs="1"/>

      <xsd:element name="length" type="xsd:string" minOccurs="1"
maxOccurs="1"/>

      <xsd:element name="flags" type="xsd:string" minOccurs="1"
maxOccurs="1"/>

      <xsd:element name="ttl" type="xsd:string" minOccurs="1" maxOccurs="1"/>

      <xsd:element name="srcIP" type="xsd:string" minOccurs="1"
maxOccurs="1"/>

      <xsd:element name="dstIP" type="xsd:string" minOccurs="1"
maxOccurs="1"/>

      <xsd:element name="type" type="xsd:string" minOccurs="1" maxOccurs="1"/>

      <xsd:element name="code" type="xsd:string" minOccurs="1" maxOccurs="1"/>

      <xsd:element name="icmpSeq" type="xsd:string" minOccurs="1"
maxOccurs="1">

      </xsd:sequence>

      <xsd:attribute name="id" type="xsd:string" use="required"/>

      <xsd:attribute name="generator" type="xsd:string" use="required"/>

      <xsd:attribute name="description" type="xsd:string" use="required"/>

      </xsd:complexType>

      </xsd:element>
                                                            </xsd:sequence>
                                                            <xsd:attribute
name="id" type="xsd:string" use="required"/>
                                                      <xsd:complexType>
                                                </xsd:element>
                                          </xsd:sequence>
                                    </xsd:complexType>
                              </xsd:element>
                              <xsd:element name="tcp">
                                    <xsd:complexType>
                                          <xsd:sequence>
                                                <xsd:element name="user">
                                                      <xsd:complexType>
                                                            <xsd:sequence>
```

```
                                                                    <xsd:element
name="event">

       <xsd:complexType>

       <xsd:sequence>

       <xsd:element name="times" type="xsd:string" minOccurs="1"
maxOccurs="1"/>

       <xsd:element name="lastDay" type="xsd:string" minOccurs="1"
maxOccurs="1"/>

       <xsd:element name="lastHour" type="xsd:string" minOccurs="1"
maxOccurs="1"/>

       <xsd:element name="firstDay" type="xsd:string" minOccurs="1"
maxOccurs="1"/>

       <xsd:element name="firstHour" type="xsd:string" minOccurs="1"
maxOccurs="1"/>

       <xsd:element ref="frecuency"/>

       <xsd:element ref="minInterval"/>

       <xsd:element name="tos" type="xsd:string" minOccurs="1" maxOccurs="1"/>

       <xsd:element name="length" type="xsd:string" minOccurs="1"
maxOccurs="1"/>

       <xsd:element name="flags" type="xsd:string" minOccurs="1"
maxOccurs="1"/>

       <xsd:element name="ttl" type="xsd:string" minOccurs="1" maxOccurs="1"/>

       <xsd:element name="srcIP" type="xsd:string" minOccurs="1"
maxOccurs="1"/>

       <xsd:element name="dstIP" type="xsd:string" minOccurs="1"
maxOccurs="1"/>

       <xsd:element name="srcPort" type="xsd:string" minOccurs="1"
maxOccurs="1"/>

       <xsd:element name="dstPort" type="xsd:string" minOccurs="1"
maxOccurs="1"/>

       <xsd:element name="seq" type="xsd:string" minOccurs="1" maxOccurs="1"/>

       <xsd:element name="ack" type="xsd:string" minOccurs="1" maxOccurs="1"/>

       <xsd:element name="tcpFlags" type="xsd:string" minOccurs="1"
maxOccurs="1"/>

       <xsd:element name="window" type="xsd:string" minOccurs="1"
maxOccurs="1"/>

       </xsd:sequence>

       <xsd:attribute name="id" type="xsd:string" use="required"/>

       </xsd:complexType>

       </xsd:element>
                                                          </xsd:sequence>
                                                          <xsd:attribute
name="id" type="xsd:string" use="required"/>
                                                 </xsd:complexType>
                                            </xsd:element>
```

```
                                        </xsd:sequence>
                                  </xsd:complexType>
                            </xsd:element>
                            <xsd:element name="udp">
                                  <xsd:complexType>
                                        <xsd:sequence>
                                              <xsd:element name="user">
                                                    <xsd:complexType>
                                                          <xsd:sequence>
                                                                <xsd:element
name="event">

      <xsd:complexType>

      <xsd:sequence>

      <xsd:element name="times" type="xsd:string" minOccurs="1"
maxOccurs="1"/>

      <xsd:element name="lastDay" type="xsd:string" minOccurs="1"
maxOccurs="1"/>

      <xsd:element name="lastHour" type="xsd:string" minOccurs="1"
maxOccurs="1"/>

      <xsd:element name="firstDay" type="xsd:string" minOccurs="1"
maxOccurs="1"/>

      <xsd:element name="firstHour" type="xsd:string" minOccurs="1"
maxOccurs="1"/>

      <xsd:element ref="frecuency"/>

      <xsd:element ref="minInterval"/>

      <xsd:element name="tos" type="xsd:string" minOccurs="1" maxOccurs="1"/>

      <xsd:element name="length" type="xsd:string" minOccurs="1"
maxOccurs="1"/>

      <xsd:element name="flags" type="xsd:string" minOccurs="1"
maxOccurs="1"/>

      <xsd:element name="ttl" type="xsd:string" minOccurs="1" maxOccurs="1"/>

      <xsd:element name="srcIP" type="xsd:string" minOccurs="1"
maxOccurs="1"/>

      <xsd:element name="dstIP" type="xsd:string" minOccurs="1"
maxOccurs="1"/>

      <xsd:element name="srcPort" type="xsd:string" minOccurs="1"
maxOccurs="1"/>

      <xsd:element name="dstPort" type="xsd:string" minOccurs="1"
maxOccurs="1"/>

      <xsd:element name="udpLength" type="xsd:string" minOccurs="1"
maxOccurs="1"/>

      </xsd:sequence>

      <xsd:attribute name="id" type="xsd:string" use="required"/>

      <xsd:attribute name="generator" type="xsd:string" use="required"/>

      <xsd:attribute name="description" type="xsd:string" use="required"/>

      </xsd:complexType>
```

```
      </xsd:element>
                                                                  </xsd:sequence>
                                                                  <xsd:attribute
name="id" type="xsd:string" use="required"/>
                                                              </xsd:complexType>
                                                        </xsd:element>
                                                  </xsd:sequence>
                                            </xsd:complexType>
                                      </xsd:element>
                               </xsd:sequence>
                        <xsd:complexType>
      </xsd:element>
</xsd:schema>
```

Schema de Registro de Reglas: Rules.xsd

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:element name="times"  type="myInteger"/>
      <xsd:element name="interval" type="xsd:decimal"/>

      <xsd:element name="minInterval">
            <xsd:complexType>
                  <xsd:sequence>
                        <xsd:element ref="times" minOccurs="1"
maxOccurs="1"/>
                        <xsd:element ref="interval" minOccurs="1"
maxOccurs="1"/>
                  </xsd:sequence>
            </xsd:complexType>
      </xsd:element>
      <xsd:element name="frecuency">
            <xsd:complexType>
                  <xsd:sequence>
                        <xsd:element ref="times" minOccurs="1"
maxOccurs="1"/>
                        <xsd:element ref="interval" minOccurs="1"
maxOccurs="1"/>
                  </xsd:sequence>
            </xsd:complexType>
      </xsd:element>
      <xsd:element name="IcmpInfo">
            <xsd:complexType>
                  <xsd:element name="type" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
                  <xsd:element name="code" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
                  <xsd:element name="icmpSeq" type="xsd:string" minOccurs="1"
maxOccurs="1">
            </xsd:complexType>
      </xsd:element>
      <xsd:element name="TcpInfo">
            <xsd:complexType>
                  <xsd:element name="srcPort" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
                  <xsd:element name="dstPort" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
                  <xsd:element name="seq" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
                  <xsd:element name="ack" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
                  <xsd:element name="flagsTcp" type="xsd:string"
minOccurs="1" maxOccurs="1"/>
                  <xsd:element name="window" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
            </xsd:complexType>
```

```
	</xsd:element>
		<xsd:element name="UdpInfo">
		<xsd:complexType>
			<xsd:element name="srcPort" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
			<xsd:element name="dstPort" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
			<xsd:element name="udpLength" type="xsd:string"
minOccurs="1" maxOccurs="1"/>
		</xsd:complexType>
	</xsd:element>

	<xsd:element name="IpInfo">
		<xsd:complexType>
			<xsd:sequence>
				<xsd:element name="tos" type="xsd:string"
minOccurs="1" maxOccurs="1"/>
				<xsd:element name="length" type="xsd:string"
minOccurs="1" maxOccurs="1"/>
				<xsd:element name="dontFrag" type="xsd:string"
minOccurs="1" maxOccurs="1"/>
				<xsd:element name="moreFrag" type="xsd:string"
minOccurs="1" maxOccurs="1"/>
				<xsd:element name="ttl" type="xsd:string"
minOccurs="1" maxOccurs="1"/>
				<xsd:element name="srcIP" type="xsd:string"
minOccurs="1" maxOccurs="1"/>
				<xsd:element name="dstIP" type="xsd:string"
minOccurs="1" maxOccurs="1"/>
				<xsd:element name="content" type="xsd:string"
minOccurs="1" maxOccurs="1"/>
			</xsd:sequence>
			<xsd:choice>
				<xsd:element ref="IcmpInfo" minOccurs="0"
maxOccurs="1">
				<xsd:element ref="TcpInfo" minOccurs="0"
maxOccurs="1">
				<xsd:element ref="UdpInfo" minOccurs="0"
maxOccurs="1">
			</xsd:choice>
		</xsd:complexType>
		<xsd:attribute name="protocol" type="xsd:string"/>
	</xsd:element>

	<xsd:element name="Rules">
		<xsd:complexType>
			<xsd:element name="rule">
				<xsd:complexType>
					<xsd:sequence>
						<xsd:element name="event">
							<xsd:complexType>
								<xsd:choice>
									<xsd:element
ref="frecuency" minOccurs="1" maxOccurs="1"/>
									<xsd:element
ref="minInterval" minOccurs="1" maxOccurs="1"/>
								</xsd:choice>
								<xsd:sequence>
									<xsd:element
ref="IpInfo" minOccurs="0" maxOccurs="1"/>
								</xsd:sequence>
								<xsd:attribute name="id"
type="xsd:string" use="required"/>
								<xsd:attribute name="type"
type="xsd:string" use="required"/>
								<xsd:attribute
name="generator" type="xsd:string" use="required"/>
							</xsd:complexType>
						</xsd:element>
					</xsd:sequence>
```

```
                                    <xsd:attribute name="action"
type="xsd:string"/>
                                    <xsd:attribute name="msg" type="xsd:string"/>
                                    <xsd:attribute name="priority"
type="xsd:string"/>
                            </xsd:complexType>
                        </xsd:element>
                </xsd:complexType>
        </xsd:element>
</xsd:schema>
```

Apéndice B: guía de instalación

REQUISITOS DE INSTALACIÓN:

Antes de instalar el monitor de acciones en un PC debemos asegurarnos que nuestro sistema cumple con los siguientes requisitos:

- Sistema operativo Windows 2000 o Windows XP.
- Espacio mínimo en disco: 1,5 MB
- Espacio en disco recomendado: a partir de 2 MB para almacenar los logs y los registros de eventos.

INSTALACIÓN:

Para poder ejecutar el monitor de acciones en el PC en un entorno Windows es necesario seguir los siguientes pasos:

INSTALACIÓN DE JAVA:

Para ello es necesario descargar e instalar la última versión del JDK de java.

Podemos hacer la descarga desde la web de descargas de java: http://java.sun.com/javase/downloads/index.jsp

Una vez descargado el instalable, se deberá lanzar la instalación.

Tras haber instalado el JDK de Java en nuestro PC es necesario modificar el valor de las variables de entorno ***"path" y "classpath".***

INSTALACIÓN DE JPCAP

Los pasos para poder instalar Jpcap en nuestra máquina se indican en el enlace de Jpcap http://netresearch.ics.uci.edu/kfujii/jpcap/doc/install.html. Aún así, se detallará todo el proceso a continuación:

- Se ha de instalar la última versión aplicación WinPcap que es la que permite escuchar y detectar el tráfico de red que llega al sistema. Podemos descargar el instalador de Winpcap desde su web: http://winpcap.polito.it/

- Una vez instalado WinPcap descargaremos el instalador de Jpcap para Windows: http://netresearch.ics.uci.edu/kfujii/jpcap/doc/download.html

- Si se tiene una versión previa de Jpcap instalada, antes de instalar la nueva versión se debe de desinstalar.

- Si se olvida de desinstalar la versión previa de Jpcap, se debe buscar los archivos "**Jpcap.dll**" y "**jpcap.jar**" en el sistema y borrarlos.

INSTALACIÓN DE LA LIBRERÍA KXML

Instalaremos la librería KXML descargando el archivo jar de la página de KXML: http://kxml.sourceforge.net/

Una vez descargado la librería es necesaria añadirla en las variables de entorno ***"path" y "classpath".***

EJECUCIÓN DEL MONITOR DE ACCIONES:

Para ejecutar el monitor de acciones, debemos aseguraros de que tenemos:

- Una carpeta llamada `classes` que contiene todos los archivos `.class` de la aplicación. Estos archivos se estructuran en carpetas que corresponden con los nombres de los paquetes definidos.
- El archivo `monitor.jar` situado en la carpeta `classes`.
- Una carpeta llamada librerías que contenga todas la librería `kxml2-2.3.0.jar`
- Una carpeta `log` para almacenar los resultados del monitor.
- Una carpeta `files` que contendrá todos los archivos XML utilizados en la aplicación junto con sus esquemas.
- El fichero de log de la aplicación WSFEP.
- Una carpeta `images` con las imágenes de la aplicación.
- Un archivo `properties.txt` en el que escribiremos el valor de las propiedades del sistema. Estas propiedades son:

    - `ptmPath`: ruta en la que se encuentra el fichero de log de WSFEP.
    - logDirectory: directorio en el que se almacenarán los ficheros de log.
    - AplRegPath: ruta del fichero de registro de eventos de aplicación.
    - NetworkRegisterPath: localización del fichero de registro de eventos de red.
    - RulesPath: ruta en la que se encuentra en fichero de reglas.
    - PotencialAtacks: indica la localización del fichero de ataques potenciales.
    - Images: localización de la carpeta con las imágenes de la aplicación.
    - user: indica id del usuario que ejecuta el monitor de acciones.

Una vez que nos hemos asegurado de que tenemos todos los ficheros y carpetas anteriores, abriremos una línea de comandos *cmd* y en la ruta en la que se encuentra la carpeta con las clases, ejecutaremos el comando:

```
java -cp ..\librerias\kxml2-2.3.0.jar;monitor.jar;. es.uc3m.it.monitor.GUI.MonitorGUI
```

Si no se quiere hacer por línea de comandos también se puede ejecutar el archivo `ejecutarMonitorGUI.bat` adjunto con la aplicación.