Univerdad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación
Departamento de Ingeniería de Sistemas Telemáticos

PROYECTO FIN DE CARRERA

Desarrollo de Mecanismos Avanzados de Supervisión y Análisis de Tráfico sobre Redes IP

Autor: Fermín Galán Márquez
Tutor: Manuel Álvarez-Campana Fernández Corredor

Julio de 2002

Proyecto Fin de Carrera

Título:	Desarrollo de Mecanismos Avanzados de Supervisión y Análisis de Tráfico sobre Redes IP
Alumno:	D. Fermín Galán Márquez
Tutor:	D. Manuel Álvarez-Campana Fernández Corredor
Departamento:	Ingeniería de Sistemas Telemáticos

Miembros del tribunal calificador

Presidente:	D. Julio José Berrocal Colmenarejo
Vocal:	D. Manuel Álvarez-Campana Fernández Corredor
Secretario:	D. Víctor Abraham Villagrá González
Suplente:	D. Enrique Vázquez Gallo

Fecha de lectura:

Calificación:

Univerdad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación
Departamento de Ingeniería de Sistemas Telemáticos

PROYECTO FIN DE CARRERA

Desarrollo de Mecanismos Avanzados de Supervisión y Análisis de Tráfico sobre Redes IP

Autor: Fermín Galán Márquez
Tutor: Manuel Álvarez-Campana Fernández Corredor

Julio de 2002

Resumen del proyecto

El presente proyecto desarrolla una plataforma de captura y análisis de tráfico para redes IP. El objetivo del trabajo es la realización de un conjunto de nuevas técnicas y mejoras para incrementar la funcionalidad de la plataforma del proyecto MIRA, preexistente al comienzo de la actividad.

En concreto, se abordan mejoras en tres grandes áreas: detección de tráfico lúdico, incorporación de funciones de seguridad y desarrollo de módulos para la generación de estadísticas convencionales. También se describen un conjunto de mejoras menores que incrementan la eficiencia y flexibilidad de la plataforma.

La plataforma ha sido probada y validada utilizando la red académica española y de investigación RedIris, si bien el desarrollo realizado permite su aplicación a múltiples escenarios.

Palabras clave

Internet, RedIris, captura y análisis de tráfico, monitorización pasiva, audio MP3, P2P, seguridad, NIDS, Snort, estadísticas convencionales.

A mis padres,
Manoli y Fermín.

A mis tíos,
Amparo y Manolo.

Este Proyecto Fin de Carrera ha sido realizado
con el apoyo de la Cátedra Amena.

Agradecimientos

El presente documento pone punto y final a cinco años de carrera. El viaje no hubiera sido posible sin la presencia de muchas personas a las que ahora debo agradecer su ayuda.

En primer lugar, al personal docente de la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universidad Politécnica de Madrid y, en especial, a Ana y Jorge en los laboratorios del DIT por el apoyo prestado durante la realización del proyecto fin de carrera. También quiero agradecer al resto del equipo del proyecto MIRA, tanto en la Universidad Carlos III de Madrid como en la Universidad Politécnica de Cataluña, su satisfactoria colaboración.

Gracias a Ricardo por atender mis dudas sobre LaTeX. A Antonio, por su ayuda con el diagrama UML. A Beto, por su asistencia en temas lingüísticos. A Fraga (auténtico pionero en eso del MP3), sin cuya colección de música mi estudio no habría podido ser tan profundo como al final ha sido.

Gracias también a Freddie Mercury, cuya voz ha sido mi única compañía en innumerables noches de insomne estudio a lo largo de estos cinco años. God save the Queen.

Mis agradecimientos también a Internet por fascinarme en noviembre de 1995. La verdadera razón de que empezara esta carrera.

Por último (pero no por ello menos importante, aunque sea un tópico) gracias a mis amigos, a los que no nombro porque sería imperdonable olvidarme de ninguno. Ellos me han enseñado cosas que no se aprenden en ningún aula. Quiero agradecerles todos los buenos momentos, pasados, presentes y futuros.

Índice General

Índice de Figuras

Índice de Tablas

Parte I

Memoria

Capítulo 1

Introducción

1.1 Contexto del proyecto

En la actualidad, Internet se ha consolidado indiscutiblemente como la red telemática más extendida y utilizada a nivel mundial. Surgida a partir de una iniciativa del Departamento de Defensa de los Estados Unidos a finales de los años 60 y desarrollada durante sus primeras décadas de vida dentro del entorno académico proporcionado por las universidades y los centros de investigación, en los últimos años ha supuesto una verdadera revolución en todos los ámbitos, no solo en el tecnológico.

En el ámbito económico, ha dado lugar a una nueva manera de entender los negocios (la llamada *economía digital*), donde la información y el *know-how* son activos con valor propio y la externalidad de la red (poder llegar a cuantos más usuarios mejor) es clave. En el ámbito social, Internet ha dejado de ser un experimento de ingenieros e informáticos para alcanzar a la sociedad en su conjunto. En el ámbito político también ha tenido un gran impacto, ya que en Internet las fronteras entre estados se disuelven y los reguladores tienen ante sí un difícil reto.

El crecimiento experimentado por esta Red de Redes ha sido espectacular. El número de usuarios conectados y el ancho de banda demandado ha crecido exponencialmente, siendo conocida la ley de Sidgmore al respecto, que establece que "la capacidad de los enlaces troncales de Internet se dobla cada tres meses y medio"[1]. El tiempo de conexión, muy importante antiguamente, cuando se utilizaba para el dimensionamiento de las redes, empieza a carecer de sentido cuando la tecnología (Ethernet, ADSL, GPRS, ...) permite ya que los usuarios estén permanentemente conectados.

[1] John Sidgmore es alto directivo de MCI/WorldCom y CEO de UUNet. El irónico corolario de esta ley es "si no le asusta es que no lo ha entendido bien".

Todo este crecimiento ha producido una auténtica eclosión de servicios y aplicaciones. En la actualidad el multimedia o el comercio electrónico no son sino la punta del iceberg de lo que ha de venir. Cuando la convergencia dé sus primeros frutos (el *web-TV* o Internet a través del teléfono móvil) y se consoliden tecnologías que ahora están en fase de investigación (como el *multicast* o la garantía de calidad de servicio) tendremos un nuevo mundo lleno de posibilidades.

No solo nos encontramos con un conjunto de nuevos servicios, sino con el surgimiento de nuevos paradigmas de comunicación. La antigua arquitectura cliente-servidor, heredera de un tiempo en el que la capacidad de proceso de los equipos era un recurso muy costoso e indiscutible hasta hace poco, ha visto surgir la aparición de un nuevo modelo de comunicación, que no necesita de servidores, ya que los intercambios se realizan "igual a igual" (*peer to peer* o P2P) entre clientes. Puesto que son los usuarios los que se interconectan directamente, el servidor central pierde importancia y controlar a los clientes y la información que intercambian es mucho más complejo. Los sistemas P2P han demostrado gran aplicabilidad para el intercambio de contenidos personales (audio, vídeo o cualquier contenido en general), rozando e incluso traspasando los límites de la legalidad en muchos casos (dado los derechos de autor que frecuentemente poseen estos contenidos).

De todo lo anterior se deduce que el tráfico que circula por las autopistas de la información de Internet es de naturaleza muy diversa. Es por tanto necesario conocer como es y como evoluciona utilizando algún tipo de plataforma de captura y monitorización (al igual que la Dirección General de Tráfico necesita conocer que flujo de vehículos y de que tipo —motocicletas, turismos, camiones, ...— circula por la red de carreteras) para poder realizar un dimensionamiento y mantenimiento adecuado de los recursos de red.

El presente proyecto fin de carrera desarrolla una de tales plataformas para la captura y análisis de tráfico, incorporando funciones avanzadas a un modelo preexistente dentro del proyecto MIRA. Vista la heterogeneidad existente en el campo de las redes telemáticas, se ha buscado principalmente la flexibilidad: que la plataforma sea adaptable al mayor número de situaciones posibles.

Si bien las herramientas de análisis de tráfico existen prácticamente desde el surgimiento de las redes telemáticas, la aquí desarrollada presenta un aspecto novedoso en su filosofía. Mientras que la mayor parte de las plataformas de captura y monitorización analizan aspectos formales del tráfico (que direcciones IP origen y destino son más frecuentes, que puertos de transporte TCP y UDP se utilizan, ...), la plataforma del proyecto MIRA inspecciona el contenido del tráfico. Se trata no de ver como son los camiones que circulan por la autopista, sino que carga transportan.

La plataforma ha sido ampliamente probada y utilizada en los enlaces troncales de RedIris (descrita en 1.3), la Red Académica Española que interconecta las distintas universidades y centros de investigación del país y uno de los mejores escenarios en los que puede verificarse la eficiencia de las metodologías de captura y monitorización implementadas.

La plataforma considera también el problema de la seguridad en Internet, cuya importancia crece cada día. Cuando se diseñó Internet se hizo como una red completamente abierta a la que, en principio, cualquiera puede conectarse. Aunque este es uno de los factores que ha permitido el vertiginoso crecimiento del que hemos hablado, implica que cualquier agente malintencionado puede aprovechar las vulnerabilidades de los hosts conectados para cometer acciones ilegales (como pueda ser el robo de datos bancarios o el espionaje industrial). Si bien antiguamente tales actividades eran más o menos aisladas (aunque con un fuerte impacto en la prensa) hoy en día, debido a la globalización de la información, no hace falta ser un gurú en sistemas operativos y redes para realizar ataques por Internet: basta con descargar herramientas especializadas en ataques en los foros que utilizan los *hackers* y utilizarlas, sin tener demasiado conocimiento de como funcionan internamente.

Como puede deducirse, el problema es lo suficientemente importante como para que estemos interesados en la incorporación de mecanismos para detectar y registrar ataques de seguridad dentro de la plataforma de captura y monitorización desarrollada.

1.2 Estructura del documento

La presente memoria se estructura en una serie de capítulos. El presente apartado forma parte del primer capítulo, junto con una descripción del contexto del proyecto y una breve descripción de RedIris.

El segundo capítulo es una descripción detallada de la plataforma de captura y análisis desarrollada previamente a la realización de este trabajo. Puesto que este documento describe un conjunto de mejoras sobre un sistema preexistente, conviene describirlo lo mejor posible, para establecer el marco de trabajo en el que se desarrollan el resto de capítulos.

El tercer capítulo entra dentro del trabajo en sí, con la descripción de técnicas específicas para la detección de tráfico lúdico, en concreto especializándose en audio MP3.

El cuarto capítulo aborda el tema de la seguridad, con la descripción de un conjunto de funciones que permiten la detección y registro de actividad sospechosa en la red monitorizada.

El quinto capítulo describe el desarrollo de un módulo para la generación de estadísticas convencionales, que complementan a las estadísticas habituales de la plataforma basadas en inspección de contenidos.

El sexto capítulo describe un conjunto de mejoras de menor envergadura, pero también importantes dentro del trabajo realizado.

Por último, en el séptimo capítulo se exponen las conclusiones del trabajo realizado, así como posibles trabajos futuros dentro de la línea de investigación aquí desarrollada.

La memoria finaliza con un conjunto de apéndices. El apéndice A describe los formatos de los ficheros de datos utilizados en la plataforma. El apéndice B contiene las interfaces de llamada que usan los programas desarrollados en el proyecto fin de carrera. El apéndice C consta de los *scripts* utilizados para la consolidación en el capítulo sobre seguridad. El apéndice D describe el algoritmo de búsqueda de patrones de Boyer-Moore. Por último, el apéndice E es un glosario en el que se describen las distintas siglas y acrónimos utilizados a lo largo del documento.

La memoria viene acompañada de un pliego de condiciones y un presupuesto del proyecto, situados al final del documento.

1.3 La Red Académica Española

Puesto que el principal caso de aplicación a lo largo de la vida del proyecto MIRA ha sido la monitorización del tráfico en la Red Académica Española ([RedIris]), se realizará a continuación una breve descripción de esta red académica de investigación.

El germen de la actual RedIris se encuentra en el programa IRIS, que nació en 1988 dentro del Plan Nacional de Investigación y Desarrollo, con el objetivo de realizar la interconexión de los recursos informáticos de las distintas universidades y centros de investigación del país. En 1991 se transforma en la actual RedIris, que desde 1994 está gestionada por el Consejo Superior de Investigaciones Científicas (CSIC).

Hoy en día, RedIris constituye la principal herramienta del Programa Nacional de Aplicaciones y Servicios Telemáticos y asume la responsabilidad de la provisión de los servicios y la infraestructura necesarios para el cumplimiento de los objetivos de dicho programa. Actualmente, da acceso a más de 260 organizaciones, principalmente universidades y centros de I+D.

RedIris participa activamente en proyectos de investigación internacionales, como son el proyecto Gèant para la constitución de una red paneuropea de investigación de banda ancha (enlaces troncales de 10 Gbps de capacidad), a la que se conecta con un enlace de 2.5 Gbps; el proyecto Internet2 (Abilene, ESnet) o proyectos de despliegue de redes avanzadas de *multicast*.

El *backbone* de la red de transporte de RedIris (figura 1.1) presenta una

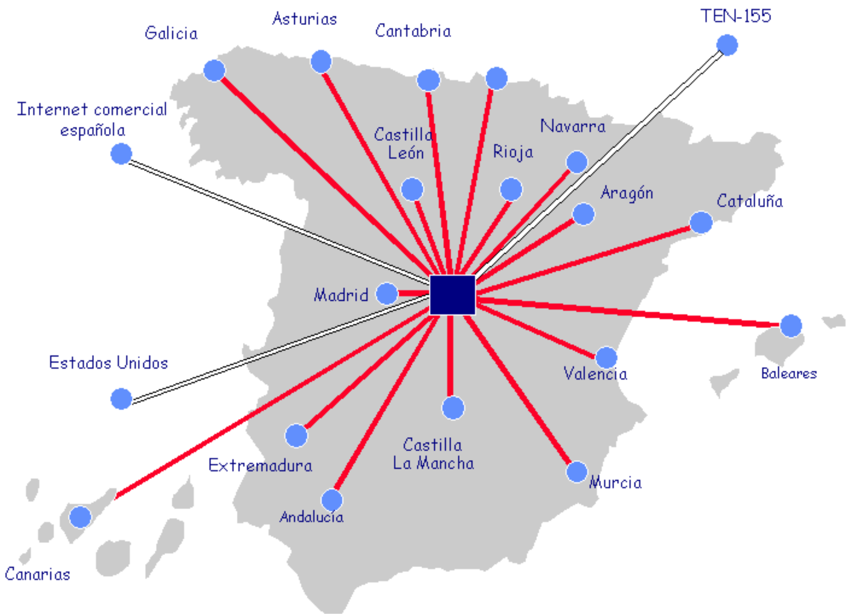


Figura 1.1: Topología de [RedIris].

topología sideral (con forma de estrella). Existen 17 nodos regionales, cada uno de los cuales concentra el tráfico de una Comunidad Autónoma. Un nodo representa un conjunto de equipos de comunicaciones que permiten la concentración de los medios de transmisión troncales y de las líneas de acceso de los centros de cada Comunidad. Todos estos nodos están conectados con un router central en Madrid mediante circuitos ATM, utilizando para ello las infraestructuras proporcionadas por la red Gigacom de Telefónica[2].

En el nodo central se encuentran también las conexiones con las redes exteriores, en concreto:

- Internet comercial española NURIA (antigua Ibernet), en el punto neutro de interconexión ESPANIX, con una capacidad de 18 Mbps a nivel IP aproximadamente.

- Red TEN-155. Se trata de una red paneuropea de investigación cuyos enlaces troncales son de 155 Mbps. La conexión de RedIris con esta red se realiza a 45 Mbps.

[2]La única excepción es el nodo de Madrid, que por encontrarse físicamente en el mismo lugar que el router, no utiliza Gigacom.

- Estados Unidos. A través de un enlace ATM de 19.2 Mbps (aproximadamente 16 Mbps IP), vía MCI.

Todo el tráfico con origen o destino en una de las Comunidades Autónomas pasa por el router central, el cual lo encaminará convenientemente bien hacia otra Comunidad Autónoma o bien hacia el exterior.

Capítulo 2

Descripción de la plataforma

2.1 Introducción

Las actividades que forman parte del presente proyecto fin de carrera han consistido en una serie de modificaciones y mejoras sobre la plataforma MIRA. En este capítulo realizaremos una descripción detallada de dicha plataforma.

El objetivo que se persigue es que al finalizar este capítulo el lector tenga una visión clara de lo que es la plataforma MIRA y en que estado se encontraba en el momento de comenzar las actividades del proyecto. De esta forma, se establece el marco de referencia en el que se desarrollarán los siguientes capítulos de esta memoria.

En primer lugar se realizará una introducción al proyecto MIRA (apartado 2.2), para pasar a continuación a una detallada descripción de su arquitectura (apartado 2.3), considerando aspectos procesales y estructurales. Por último, se describirán cuales son los escenarios de aplicación en los cuales la plataforma puede ser usada (apartado 2.4), tratando con especial interés el despliegue realizado en la Red Académica Española RedIris, ya que es donde se han realizado la mayor parte de las pruebas reales durante el tiempo de realización del proyecto.

2.2 Proyecto MIRA

El proyecto MIRA[1] es continuación de los proyectos CASTBA[2] y MEHARI[3], desarrollados dentro del Plan Nacional de I+D, constituyendo en con-

[1] Metodologías para la Inspección de Redes Avanzadas.

[2] CAlidad de los Servicios Telemáticos provistos por la red académica de Banda Ancha.

[3] MEcanismos y Herramientas para el Análisis del uso de servicios Internet aplicado a RedIris.

junto una línea de investigación sobre caracterización y análisis del tráfico de RedIris de cuatro años de duración. Puede obtenerse información más concreta sobre CASTBA y sus resultados en [DIT 96], [Pérez98], [Álvarez+ 98] y [Álvarez+ 99a], así como de MEHARI en [DIT 97], [DIT 98a], [DIT 98b], [Álvarez+ 99b], [Lizcano+ 99] y [Azcorra+ 99].

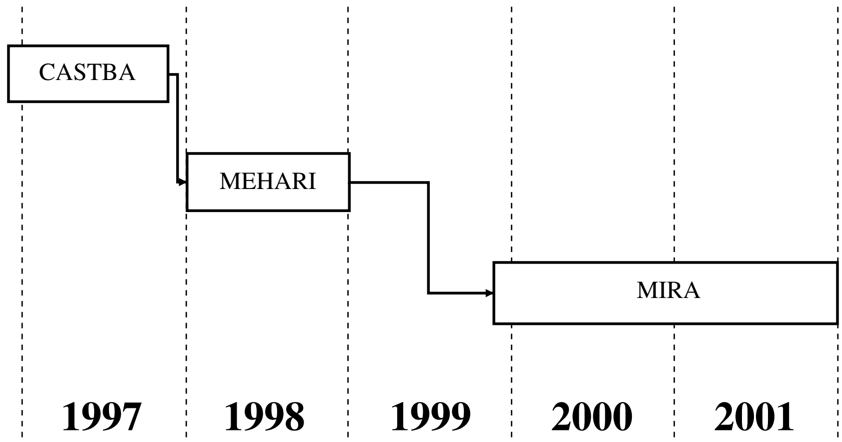


Figura 2.1: Proyectos CASTBA, MEHARI y MIRA.

En lo que al MIRA se refiere, es un proyecto CICYT (2fd-97-2234-c03-01) cofinanciado con fondos FEDER. La duración del proyecto estuvo comprendida entre el 30 de diciembre de 1999 y el 31 de diciembre de 2001. Los participantes de este proyecto fueron la Universidad Carlos III de Madrid, la Universidad Politécnica de Madrid, la Universidad Politécnica de Cataluña, Telefónica I+D y RedIris.

El objetivo del proyecto MIRA es el desarrollo de una plataforma avanzada de supervisión y control de tráfico para redes académicas y de investigación. El punto de partida es el prototipo experimental MEHARI, desarrollado por los mismos grupos investigadores que participaron en la herramienta MIRA.

En la figura 2.2 se muestra una descomposición de los distintos paquetes de trabajo del proyecto, cada uno de los cuales se divide en una serie de actividades. Se describen aquellas actividades relacionadas con la parte de trabajo que ha sido realizada en este proyecto fin de carrera. Para información adicional sobre el resto de actividades, acudir a [Azcorra 99].

- Detección avanzada. Aunque no se consideró inicialmente como una actividad en sí dentro del plan de proyecto inicial, el crecimiento de ciertos tipos de tráfico, especialmente el de tipo lúdico, lleva a establecer como objetivo el desarrollo de técnicas y métodos avanzados que permitan su detección eficientemente.

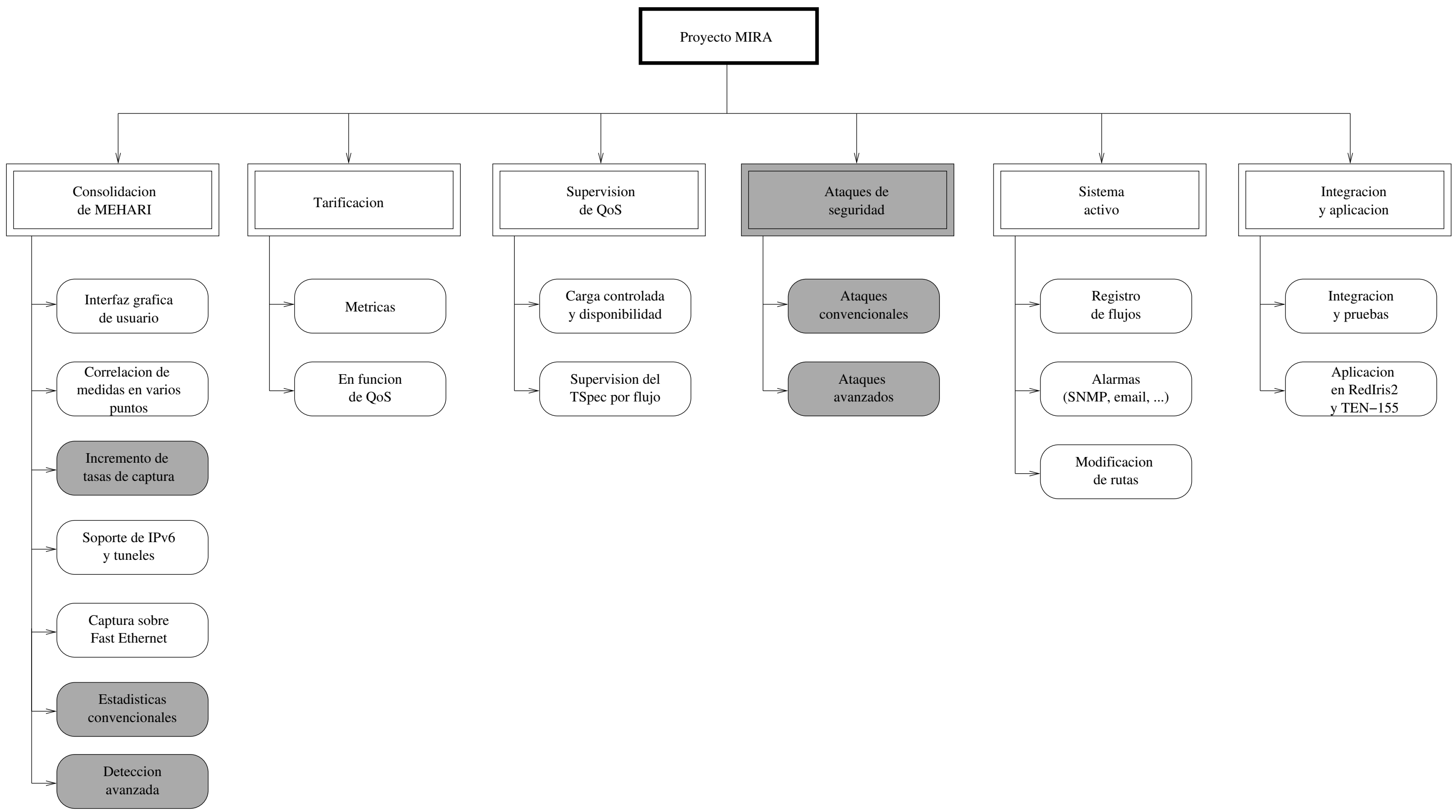


Figura 2.2: Paquetes de trabajo del proyecto MIRA.

El trabajo realizado dentro del presente proyecto fin de carrera como contribución a esta actividad se detalla en el capítulo 3.

- Ataques de seguridad. Se quiere dotar a la plataforma MIRA de un conjunto de facilidades para la detección e identificación de ataques de seguridad, tanto convencionales como avanzados. También permitiría la detección del acceso o utilización indebida de recursos o información a través de la red.

  La situación del sistema MIRA dentro de la red será, al menos en la mayoría de los casos, en puntos centrales de la misma, tales como los enlaces troncales. Debido a esta ubicación el control de la herramienta es global, sirviendo de complemento perfecto a sistemas de seguridad locales, tales como los cortafuegos. Esta ubicación permitiría, por ejemplo:

  - La identificación de ataques de seguridad indirectos, a través de más de un host, facilitando la detección del foco del ataque.
  - La localización de ataques simultáneos desde un mismo punto a varios puntos dentro de la red.
  - Inspeccionar el tráfico de manera pasiva, dificultando en gran medida el ataque al propio sistema de seguridad.

  El trabajo realizado dentro del presente proyecto fin de carrera como contribución a esta actividad se detalla en el capítulo 4.

- Estadísticas convencionales. La evocación del sistema MEHARI como plataforma avanzada para la monitorización, análisis y operación de redes académicas, sugiere la conveniencia de dotar al prototipo de un módulo de generación de estadísticas de tráfico convencionales. Dentro de estas cabe incluir estadísticas generales sobre utilización y carga de los enlaces troncales, retardos de tránsito, ratios de pérdida de paquetes, distribución de tráfico por servicios y aplicaciones (TCP/UDP, HTTP, NTP, ...), etc. Este tipo de estadísticas ya fueron consideradas dentro de la Acción Especial CASTBA, donde se llevó a cabo el desarrollo de los correspondientes módulos de análisis para su obtención de manera semiautomática. La integración de dichos módulos en el sistema MEHARI, y la posibilidad de facilitar la visualización gráfica de resultados, sería otra de las características que contribuiría a aumentar el valor añadido de la herramienta.

  El trabajo realizado dentro del presente proyecto fin de carrera como contribución a esta actividad se detalla en el capítulo 5.

- Incremento de las tasas de captura. Entre las principales virtudes del prototipo experimental MEHARI destaca su elevada capacidad de captura, cuatro veces superior a las proporcionadas por los analizadores de tráfico comerciales del momento, con un coste hardware ocho veces menor. No obstante, dicha capacidad puede no ser suficiente en función de la aplicación de análisis de tráfico que se considere. En consecuencia, dentro del objetivo genérico de consolidación del sistema MEHARI se pretende dedicar parte del esfuerzo a investigar posibles maneras de mejorar las prestaciones del sistema. La labor no se antoja fácil, puesto que implica en gran medida abordar aspectos estrechamente vinculados con el hardware de captura, cuyos detalles no son fácilmente accesibles. No obstante, la experiencia adquirida durante el desarrollo del software de captura deja abierta la posibilidad de llevar a cabo ciertas optimizaciones.

  La mejora de prestaciones también es extensible a las aplicaciones de análisis de tráfico, donde precisamente se identificó el cuello de botella en la aplicación práctica del prototipo MEHARI a la red troncal de RedIris. En consecuencia, parece conveniente revisar el software de análisis a fin de investigar posibles algoritmos que conduzcan al aumento de las prestaciones globales del sistema.

  En todo caso, cabe señalar que durante el diseño de la arquitectura MEHARI, se previó la posibilidad de incrementar las prestaciones del sistema replicando las plataformas de captura y análisis. Esta es justamente una de las actividades que se realizan dentro de este paquete de trabajo, la replicación de plataformas de captura y análisis.

  El trabajo realizado dentro del presente proyecto fin de carrera como contribución a esta actividad se detalla en los apartados 6.2 y 6.3.

Por último, mencionar que, debido a su carácter de proyecto de investigación, en el marco del proyecto MIRA se han producido varios artículos que describen las funcionalidades o los resultados de la plataforma. En particular, destacamos [Veciana+ 00] y [Romeral+ 02].

2.3 Arquitectura

El objetivo del sistema MIRA es la captura y el análisis del tráfico de la red monitorizada donde se despliega la plataforma.

A alto nivel, podemos descomponer la *plataforma* en tres subsistemas interrelacionados (figura 2.3).

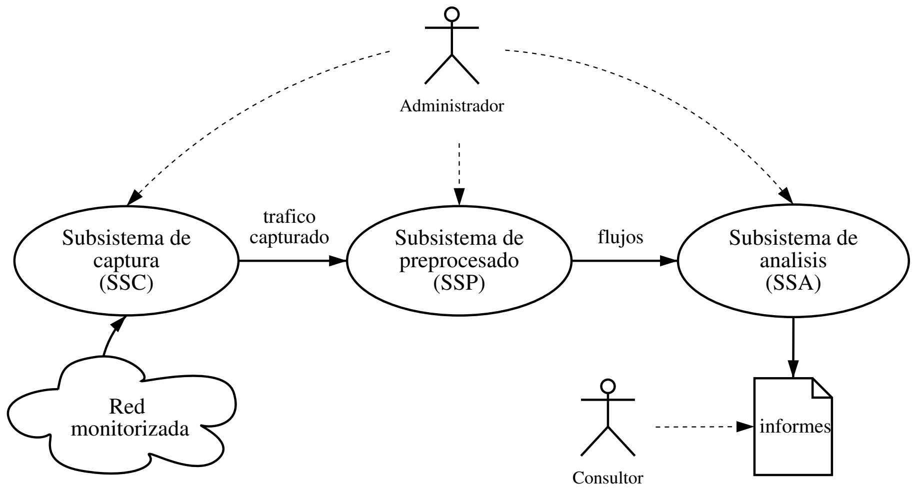


Figura 2.3: Arquitectura de la plataforma MIRA.

- Subsistema de captura (SSC). Es el responsable de la captura del tráfico en la *red monitorizada*. Los datos capturados constituyen la entrada del preprocesado (SSP).

- Subsistema de preprocesado de tráfico (SSP). Realiza un primer tratamiento del tráfico capturado por el SSC. El conjunto de paquetes se agrupa en flujos, de cada cual se extrae la información de interés que utilizará el sistema de análisis. De esta forma, se reduce el volumen de información y se aligera la carga de proceso para el subsistema de análisis[4]. Más adelante se detallará en que consiste exactamente el preprocesado.

- Subsistema de análisis (SSA). El proceso de análisis tiene como resultado la clasificación del tráfico, haciendo uso de la información proporcionada por el SSP. Los resultados son ofrecidos mediante la generación de un conjunto de informes (histogramas, gráficas, tablas, …). Más adelante se detallará en que consiste exactamente el análisis.

Dos tipos de agentes interactúan con la plataforma. El *administrador* es el encargado de configurar los distintos módulos para que se obtengan los resultados deseados, ya que una de las características más importantes de MIRA es la gran flexibilidad que permite. El *consultor* es el receptor de los

[4]Existen razones adicionales de tipo legal que obligan a este preprocesado ya que, si bien está permitido la inspección de tráfico en redes públicas (aunque las comunicaciones que por ellas transitan sean privadas) para la obtención de información estadística, no está permitido en modo alguno el registro de estas comunicaciones.

informes generados, proporcionados por el SSA. Realmente, ambos perfiles pueden coincidir en una misma persona. En el apartado 2.3.3 se explica como se realiza el acceso a la plataforma, para ambos casos.

Esta descomposición de la plataforma es a muy alto nivel y, si bien resulta interesante para describir los grandes bloques arquitecturales de la misma, es de poca utilidad a la hora de explicar aspectos concretos de funcionamiento. Por ello, en los dos siguientes apartados se realizará una descripción de los módulos procesales (apartado 2.3.1) y de la infraestructura física en la que estos se sitúan (apartado 2.3.2).

Antes de proceder con esta exposición, conviene explicar claramente el significado que haremos en lo sucesivo de alguno de los términos que hemos empleado, con el fin de evitar ambigüedades en su interpretación.

Plataforma. Nos referimos de esta forma al sistema de captura y análisis de tráfico desarrollado en el proyecto MIRA, que se despliega para la monitorización de una red.

Red monitorizada. La red en la cual se sitúa la plataforma. En el apartado 2.4 se describen posibles escenarios de aplicación.

Administrador[5]. Por administrador entendemos al usuario de la plataforma de captura y análisis, es decir, el encargado de su gestión (configuración, supervisión y control) para que produzca los resultados deseados por el consultor.

Consultor. Es el "usuario final" de la plataforma, interesado en los resultados proporcionados por la misma (principalmente, en la forma de informes). Normalmente, esta figura está asociada a la del *responsable* de la red monitorizada, término que también utilizaremos.

2.3.1 Procesos de la plataforma

Por razones de modularidad, escalabilidad y flexibilidad, se decidió dividir los distintos subsistemas en un conjunto de procesos autónomos. La forma más sencilla de entender el funcionamiento de la plataforma es considerar una *cadena de proceso* de información, desde la red monitorizada hasta los informes. Los elementos de esta cadena son procesos, cada uno de los cuales toma su entrada del proceso anterior (salvo el primero) y se la proporciona al siguiente proceso. En un principio, estos módulos trabajan concurrentemente.

[5] El término administrador puede resultar confuso, ya que puede hablarse de "administrador de la plataforma" (lo que en nuestra terminología es simplemente "administrador") y "administrador de la red" (lo que en nuestra terminología es "consultor").

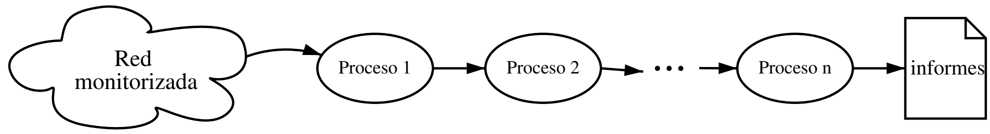


Figura 2.4: Cadena de proceso.

Para la comunicación de datos entre procesos se utiliza el paradigma de *ficheros*. La plataforma utiliza distintos tipos de ficheros, a los que se hará referencia a continuación. El formato detallado de cada uno de ellos se describe en el apéndice A.

Puede realizarse una proyección clara entre la descripción a alto nivel realizada anteriormente y la descomposición en procesos, tal y como muestra la figura 2.5. El proceso *dumpcap* se correspondería al SSC; los procesos *anaflow*, *accum*, *biflow* al SSP; y los procesos *vflow*, *mflow*, *sumrep* y los procesos del web dinámico al SSA.

$$\text{MIRA}\left\{\begin{array}{lll} \text{Subsistema de captura} & \longrightarrow & \textit{dumpcap} \\ \text{Subsistema de preprocesado} & \longrightarrow & \left\{\begin{array}{l}\textit{anaflow}\\ \textit{accum}\\ \textit{biflow}\end{array}\right. \\ \text{Subsistema de análisis} & \longrightarrow & \left\{\begin{array}{l}\textit{vflow}\\ \textit{mflow}\\ \textit{sumrep}\\ \text{procesos web dinámico}\end{array}\right. \end{array}\right.$$

Figura 2.5: Descomposición de la plataforma en procesos.

Observar que la descomposición modular permite un elevado grado de reutilización a la hora de adaptar la plataforma a distintas tecnologías de red (ATM, Ethernet, Frame Relay, ...), ya que el único proceso que habría que modificar es el primero, encargado de la captura, pues es el único que depende físicamente de la tecnología de red monitorizada empleada.

Gran parte del comportamiento de estos procesos es configurable, lo cual dota a la plataforma de una gran flexibilidad.

A continuación describiremos el trabajo concreto de cada uno de los procesos de la cadena, omitiendo los detalles sobre su implementación (como se ejecutan los procesos en los equipos, como se pasan los ficheros entre ellos, ...)

que, si bien son de importancia, serán descritos más adelante en el apartado 2.3.2.

2.3.1.1 *dumpcap*

Se encarga de capturar el tráfico de paquetes de la red monitorizada, agregarlo y proporcionárselo en ficheros CAP al proceso *anaflow*. Este proceso debe proporcionar una abstracción de la tecnología de red. Es decir, los ficheros CAP han de ser neutros con respecto al tipo de red empleada, ya sea esta ATM o Ethernet, por ejemplo[6].

Debido a lo anterior, *dumpcap* es muy dependiente de la interfaz física de captura. Entre las tareas que realiza se encuentran la comunicación con el *driver*[7] del hardware conectado a la red monitorizada, reensamblado de tramas, extracción de marcas de tiempo y agregación de paquetes para su posterior volcado en los ficheros CAP.

Dependiendo del nivel de complejidad (según la tecnología concreta de red), pueden necesitarse procesos previos a *dumpcap*, si bien esto para *anaflow* es transparente. Así ocurre en ATM, como veremos en el apartado 2.4.1.

2.3.1.2 *anaflow*

Analiza los ficheros CAP de captura que le pasa *dumpcap*, proporcionando su resultado en forma de ficheros AFL. Estos ficheros están estructurados como un conjunto de *flujos unidireccionales*. Un flujo está formado por todos aquellos datagramas que comparten una misma dirección IP origen, dirección IP destino, puerto origen y puerto destino[8].

En los ficheros AFL se recoge información sobre el número de bytes y paquetes cursados por el flujo durante un determinado periodo de tiempo (configurable). Adicionalmente, se realiza un análisis explorando el contenido de los

---

[6]Realmente, no se consigue una completa abstracción, ya que se asume que la red es orientada a conexión (por ejemplo, ATM), pues en los ficheros CAP hay un campo utilizado para indicar VPI/VCI. Aunque en el momento de diseñar la arquitectura se pensó que esto era conveniente, ya que se estaba utilizando ATM en RedIris, se plantean problemas a la hora de generalizar la arquitectura a otros tipos de redes no orientadas a conexión, como Ethernet.

[7]Por *driver* entendemos la primera capa de software por encima de un dispositivo físico, que se comunica con este mediante mecanismos hardware (interrupciones, DMA, . . .) y que constituye la interfaz de programación para aquellos procesos de usuario que quieran utilizar el dispositivo. Suele emplearse equivalentemente el término *controlador*.

[8]Adicionalmente, paquetes del mismo flujo comparten el mismo VPI/VCI. Dada la estructura sideral de RedIris (descrita en 1.3) el VPI/VCI no resulta determinante en el flujo, ya que la dirección IP origen determina unívocamente en que Comunidad Autónoma se encuentra un host y, por tanto, el VPI/VCI a ella asociado.

paquetes en busca de *patrones*. La presencia de uno de estos patrones en un paquete es lo que se denomina *síntoma*. Por ejemplo, el síntoma PUBLICIDAD puede estar indicado por la presencia del patrón 'anuncio'. Estos síntomas se cuantifican, de forma los flujos del fichero AFL incluirán una lista de todos los síntomas detectados y en que número.

Los distintos síntomas que debe considerar el motor de análisis de *anaflow*, así como de que patrones consta cada uno de ellos, son configurados por el administrador mediante un fichero que se carga en el momento de arrancar el proceso (cuyo nombre habitual es rule.cfg). En la figura 2.6 se muestra un fragmento de ejemplo de dicha configuración.

```
MAXLEN=500
MUSICA;iSTR;music
MUSICA;iSTR;rock
MUSICA;iSTR;heavy metal
PUBLICIDAD;iSTR;publicidad
PUBLICIDAD;iSTR;anuncio
JUEGOS;iSTR;game
JUEGOS;iSTR;player
JUEGOS;iSTR;juego
IRC;iSTR;irc
IRC;iSTR;nickname
IRC;iSTR;∧kick
```

Figura 2.6: Ejemplo de fichero rule.cfg.

Cada línea especifica un patrón. El primer campo (antes de ;) indica el nombre del síntoma al que pertenece el patrón, el segundo el tipo de búsqueda que se realizará para localizar el patrón (STR indica coincidencia exacta del patrón, iSTR indica coincidencia sin considerar la distinción entre mayúsculas y minúsculas) y el tercer campo el patrón en sí. Si el tercer campo empieza por circunflejo (∧) el patrón ha de buscarse tan solo al inicio del paquete.

El fichero puede contener también una directiva MAXLEN, que indica que ha de buscarse en los datos solo hasta la longitud indicada. Esto se realiza para evitar tener que recorrer el contenido del paquete entero, teniendo en cuenta que gran parte de los patrones se utilizan para detectar palabras claves de protocolos de aplicación (y por tanto, que se encuentran en la zona inicial del paquete).

2.3.1.3 *accum*

Este proceso toma como entrada distintos ficheros AFL y los *agrega* en uno único, sumando la información de todos los flujos coincidentes. Normalmente *anaflow* realiza la acumulación en un corto periodo de tiempo (del orden de segundos), siendo necesario *accum* para agregar la información en periodos más largos (horas, por ejemplo). De esta forma, se consigue un aumento de eficiencia, al reunir en un único fichero la información que estaba dispersa en varios. El periodo de agregación es configurable.

En la figura 2.7 se muestra un ejemplo de como se produce la agregación de flujos.

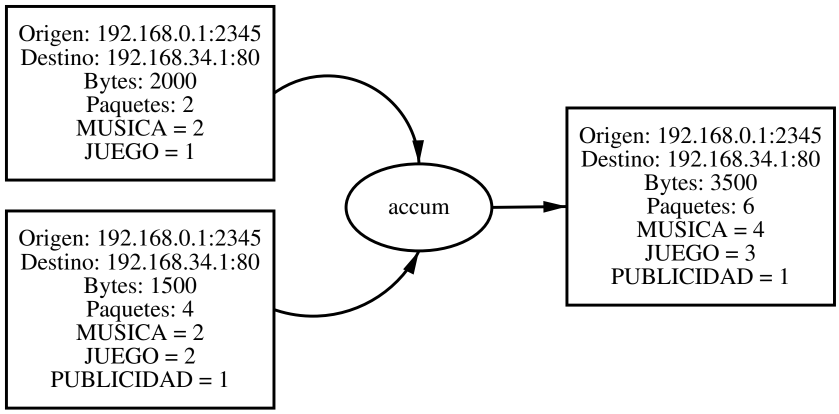


Figura 2.7: Ejemplo de agregación de flujos en *accum*.

2.3.1.4 *biflow*

Los ficheros AFL contienen información sobre flujos unidireccionales. El proceso *biflow* construye otro tipo de flujos, a partir de flujos unidireccionales de los ficheros AFL, realizando la *correlación* del tráfico que estos cursan. El correlado de tráfico resulta de gran utilidad para la clasificación posterior de tráfico que no podría ser clasificado de otro modo (aquel con pocos o ningún síntoma), por su asociación con tráfico donde si hay un gran número de síntomas. La salida de estos flujos correlados se produce en ficheros BFL.

A pesar de que podría pensarse en un proceso genérico que correlara n flujos, el caso más habitual es tener únicamente dos sentidos. Normalmente estos se corresponden al sentido de entrada y de salida de un mismo enlace de red donde se han situado las respectivas sondas de captura[9]. Por tanto se

[9] En otro tipo de escenarios, como las redes locales difusivas, resulta más complicado

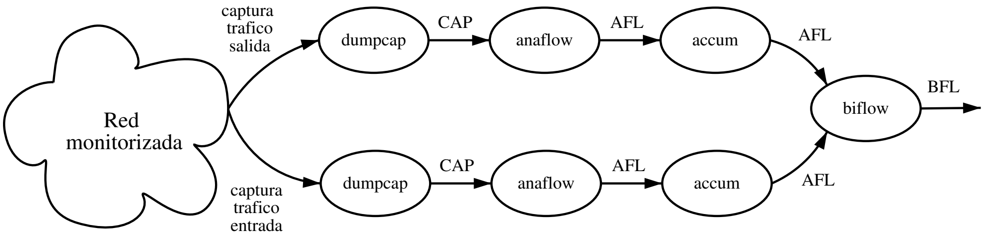


Figura 2.8: Correlación de tráfico.

consideran *flujos bidireccionales.*

Dos flujos unidireccionales se correlan cuando las direcciones IP y puertos de entrada de uno se corresponden a las direcciones IP y puertos de salida del otro y viceversa. Es decir, un flujo de entrada entre el puerto $p1$ de A y el puerto $p2$ de B se correla con un flujo de salida entre el puerto $p2$ de B y el puerto $p1$ de A. La figura 2.9 muestra un ejemplo.

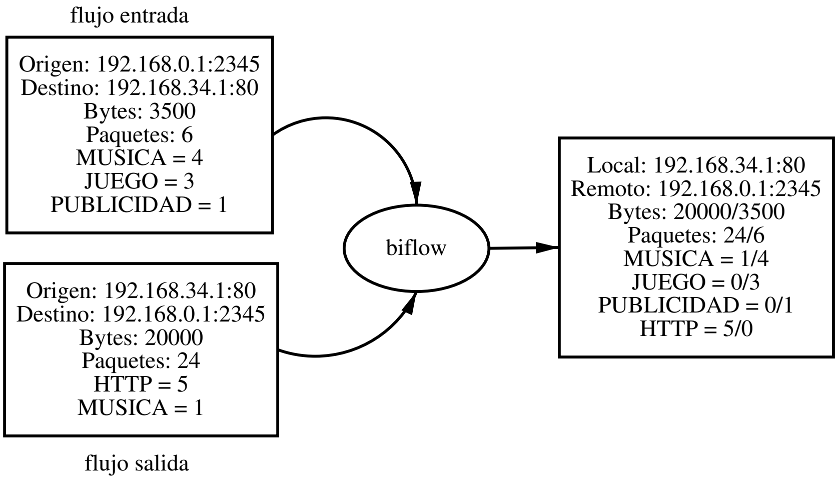


Figura 2.9: Ejemplo de correlación de flujos en *biflow.*

La mayor parte de las comunicaciones en Internet son bidireccionales, presentan una fuerte asimetría[10] y la detección de síntomas suele producirse mayoritariamente en un sentido. La correlación de tráfico tiene una gran importancia, como se apreciará en el siguiente ejemplo.

definir que se entiende por tráfico de entrada y tráfico de salida. Las soluciones pasan por introducir categorías de tráfico con una semántica equivalente a la de entrada y salida.

[10]Esto es debido normalmente al paradigma cliente-servidor: un cliente realiza una petición (pequeño tamaño) al servidor y este le devuelve un gran volumen de información.

Supongamos que un cliente hace una petición de 1 Kbyte a un servidor de ficheros de música. En esta petición se ha detectado el síntoma MUSICA repetidas veces (debido a la detección de ciertos patrones clave en las cabeceras del protocolo de la aplicación de transferencia de ficheros). El servidor devuelve como respuesta un fichero de 10000 Kbytes de música en el que no se detectan síntomas. Gracias a la correlación podremos clasificar como tráfico lúdico 10001 Kbytes de tráfico. Compárese con el 1 Kbyte que podríamos clasificar (dejando los otros 10000 como tráfico desconocido) si no realizáramos correlación de flujos.

2.3.1.5 *vflow*

Los ficheros BFL contienen un conjunto de flujos, en cada uno de los cuales aparece una lista de síntomas ponderados. El proceso *vflow* se encarga de dar un *veredicto* para cada uno de estos flujos, en función de los síntomas que presente (y, adicionalmente, direcciones IP y puertos). El objetivo es clasificar el tráfico en una de las siguientes categorías: académico (ACA), lúdico (LUD), comercial (COM) o indeterminado (IND).

```
SYM;LUD;IRC,SEX,DROGAS,JUEGOS,SPORT,MUSICA
SYM;COM;SHTTP,HTTPS,APUESTA,PUBLICIDAD, \
    CREDIT,AUTHENTICATE,BANCA
SYM;ACA;HTTP,FTP,NEWS,MAIL,TELNET,RLOGIN

REG;HTTP;80
REG;FTP;20,21
REG;NEWS;119
REG;MAIL;25
REG;TELNET;23
REG;RLOGIN;513

ESP;LUD;6667

VCC;CAL;111/111
VCC;CAT;111/112
...
```

Figura 2.10: Ejemplo de fichero verdict.cfg.

El algoritmo de determinación de veredicto se muestra en la figura 2.11. Este algoritmo, así como el número de categorías de tráfico no son configu-

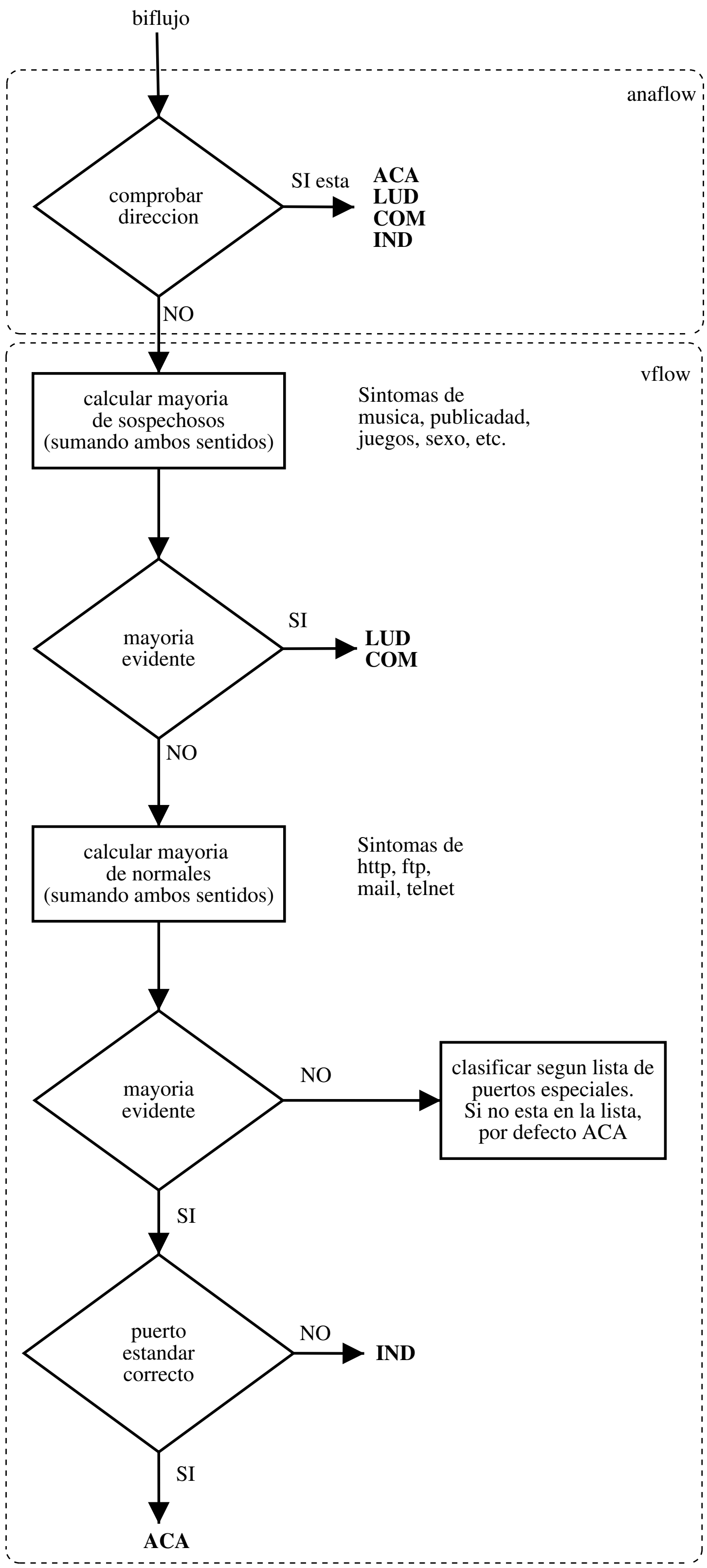


Figura 2.11: Algoritmo de determinación de veredicto.

rables (aunque se plantea como mejora su flexibilización en este sentido, ver apartado 6.4). La configuración se realiza mediante un fichero (cuyo nombre es verdict.cfg típicamente), del cual se muestra un ejemplo en la figura 2.10. Los aspectos configurables que intervienen en el proceso de decisión son[11]:

- Síntomas asociados a cada uno de los veredictos. Por ejemplo, los síntomas MUSICA o JUEGOS pueden asociarse al veredicto LUD, los síntomas PUBLICIDAD y BANCA a COM y HTTP y FTP a ACA. Especificados con 'SYM' en verdict.cfg.

- Puertos especiales. Asociados a un veredicto, debido a la presencia de un servicio claramente determinante de la categoría de tráfico que usa ese puerto. Por ejemplo, el puerto 6667 asociado a LUD (este puerto es utilizado por aplicaciones de IRC, claramente de carácter lúdico). Especificados con 'ESP' en verdict.cfg.

- Puertos estándar. Puertos de servicios estandarizados en Internet, cuyo uso no supone ningún indicio de actividad sospechosa. Se asocian con aquellos síntomas del veredicto académico. Por ejemplo HTTP 80 y FTP 20 y 21. Especificados con 'REG' en verdict.cfg.

El proceso de determinación de veredicto es el siguiente:

- El primer paso es comprobar si alguna de las dos direcciones, local o remota, se encuentra en una lista de direcciones cuyo tipo de tráfico es conocido de antemano (direcciones de proveedores de Internet, bancos, etc.). En tal caso el veredicto es el que acompaña a esa dirección.

  La lista de direcciones se especifica en un fichero de configuración (cuyo nombre típico es address.cfg) y cuyo ejemplo se muestra en la figura 2.12. Se especifican direcciones y puertos, pudiéndose usar el * como comodín.

  Aunque se describa aquí, esta parte de la determinación de veredicto no la realiza *vflow* sino *anaflow*, antes de preprocesar el flujo. De esta forma, se mejora la eficiencia de la plataforma, ya que se evita hacer la búsqueda de síntomas (computacionalmente costosa) si el flujo puede ser clasificado directamente a partir de su dirección.

[11] Observar que, adicionalmente, el fichero de configuración incluye la correspondencia entre Comunidades Autónomas y VPI/VCIs (especificada con 'VCC'). Esta información no interviene en absoluto en el proceso de decisión, se utiliza para generar los ficheros REP de salida del proceso.

```
212.106.193.97|*|LUD|tucows.ya.com|
212.35.213.254|*|COM|subastas.ya.com|
212.106.193.72|*|COM|ya.com|
212.106.193.108|*|LUD|mapas.ya.com|
193.127.102.152|*|LUD|casas.ya.com|
```

Figura 2.12: Ejemplo de fichero address.cfg.

- En el caso de no estar las direcciones en la lista se efectúa un recuento de los síntomas de tráfico lúdico o comercial (sumando los de ambos sentidos). Si el resultado muestra una mayoría evidente de tráfico lúdico o comercial (siempre que hubiera al menos un síntoma lúdico o comercial), se emite el veredicto correspondiente. Si hay empate entre ambos, se decide comercial.

- En caso contrario, se procura determinar el tipo de aplicación que está generando el tráfico del flujo. Si el recuento de síntomas no muestra una mayoría clara, se trata de clasificar el flujo en base al número de puerto utilizado (comprobando si es un puerto especial) y, en caso contrario, dando veredicto ACA por defecto.

- Si el recuento de síntomas revela una mayoría evidente y el puerto utilizado es el habitual (puerto estándar) para esa aplicación, se emite el veredicto de tráfico académico, siguiendo un criterio de presunción de inocencia. Si el puerto no es el habitual se cataloga como indeterminado (IND).

El proceso *vflow* genera dos tipos de fichero de salida.

- Ficheros VFL. Contiene el conjunto de flujos clasificados. Básicamente, la información de flujo en la misma que en un BFL, pero añadiendo la categoría de tráfico dada como veredicto.

- Ficheros REP. Este fichero consiste en una tabla donde se muestra un informe del tráfico de entrada y salida del conjunto de flujos para cada tipo de veredicto.

2.3.1.6 *mflow*

Si bien las primitivas versiones de la plataforma (CASTBA y MEHARI) consideraban el uso de un único punto de monitorización, en MIRA se plantea la posibilidad de utilizar varios de ellos. De esta forma, tenemos varias cadenas de proceso independientes que al final realizan la *consolidación* de todos los datos resultado del análisis. De esta forma, se consiguen dos ventajas significativas:

- Una mayor orientación de la plataforma hacia una arquitectura distribuida, que puede resultar muy conveniente para ciertas topologías de red. Podemos tener distintos puntos de medida en distintos lugares de la red monitorizada y consolidar la información de todos ellos.
- Aumentar la eficiencia de análisis replicando recursos de captura y análisis en un mismo punto e integrando la información al final.

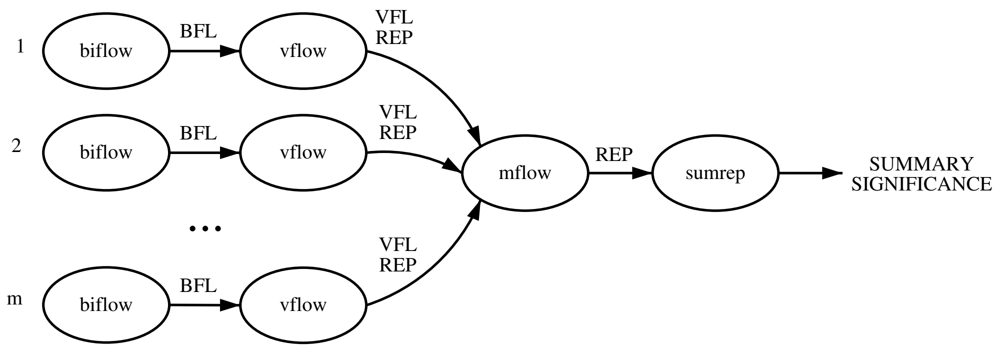


Figura 2.13: Consolidación de tráfico.

Suponiendo que tenemos m cadenas de procesado, el proceso *mflow* puede realizar dos tipos de consolidaciones, la de ficheros VFL o la de ficheros REP (la más utilizada dentro de la implantación real de la plataforma). En ambos casos se consolida el conjunto de m ficheros que arrojan como salida los respectivos *vflow*s.

- Consolidación de VFL. El fichero de salida es otro fichero VFL con el agregado de los flujos de todos los VFL de entrada. Existe una ligera modificación, pues se incluye un campo en los flujos con un número 0, 1, ..., $m-1$ que indica el origen del flujo.
- Consolidación del REP[12]. El fichero de salida es otro fichero REP que se obtiene por concatenación de todas las filas de los ficheros REP de entrada, incluyendo en cada fila un número 0, 1, ..., $m-1$ que indica

[12]Las distintas variantes de los ficheros VFL y REP se describen detalladamente en los apéndices A.5 y A.6.

su origen. Observar que no se realizan sumas de filas iguales, aunque su VPI/VCI sea el mismo, salvo la de totales (que sigue mostrando la suma de todas las filas del fichero).

2.3.1.7 *sumrep*

A pesar de que se describe aquí como proceso, *sumrep* es más una herramienta para la generación de informes que se invoca cuando es preciso. Tomando como entrada un conjunto de ficheros REP consolidados (salida de *mflow*), realiza un análisis de todos ellos. La salida es un nuevo fichero de tipo REP[13].

Adicionalmente, pueden generarse un conjunto de ficheros (identificables por la extensión .bat) que indican el tráfico de entrada y salida (en bytes y número de paquetes), así como las velocidades medias de transmisión en Mbps y kps[14].

La herramienta *sumrep* permite generar tanto informes donde se muestra el tráfico realmente capturado como estimaciones del tráfico total que circula por el enlace (basadas en proporciones, bajo la hipótesis de que el tráfico no capturado se comporta igual que el capturado). Por ejemplo, si se capturaron 10 Gbytes con una distribución 50% tráfico ACA, 25% de LUD, 15% de COM y 10% de IND y se sabe que esa cantidad es el 1% del total de tráfico, la estimación del tráfico será: 500 Gbytes de ACA, 250 Gbytes de LUD, 150 Gbytes de COM y 100 Gbytes de IND.

Esta estimación se realiza inteligentemente por periodos horarios, eliminando de la muestra aquellos intervalos en los que la plataforma ha estado caída. Junto con el fichero SUMMARY se genera un fichero SIGNIFICANCE, con información de estos periodos de actividad.

2.3.1.8 Otros procesos

En los apartados anteriores se han indicado los procesos principales que se utilizan en la plataforma, si bien no son todos los que existen.

- Procesos de gestión.
  - Los ficheros intermedios de la plataforma se usan solo para la comunicación entre procesos, por lo que se borran automáticamente. Sin embargo, los ficheros VFL y REP constituyen resultados finales, cuyo almacenamiento y borrado hay que gestionar, ya que una

[13] Aunque el convenio es que este fichero tenga por nombre SUMMARY.

[14] Miles de paquetes por segundo.

producción incontrolada de los mismos podría saturar los recursos de almacenamiento. Esta tarea de gestión se realiza a través del proceso multifunción *coll*.

La política más conveniente es almacenar los resultados comprimidos (por ejemplo, con gzip) durante un periodo razonable de tiempo (por ejemplo, un mes), tras lo cual proceder a su borrado o a la realización de *backups* externos.

- Procesos de vigilancia. La situación ideal es todo el sistema funcionando perfectamente todo el tiempo. Sin embargo, debido a la presencia de fallos ocasionales, es posible que algún proceso se caiga y la plataforma no funcione correctamente durante un periodo de tiempo. Para evitar que esto ocurra, existen procesos de vigilancia que se encargan de verificar que todo esté en orden y que han demostrado su utilidad durante la utilización de la plataforma.

  Estos procesos de vigilancia tienen la suficiente autonomía para restaurar la plataforma cuando se producen fallos menores. De encontrarse un fallo persistente, no son capaces de corregirlo y se encargan de notificar la situación anómala al administrador para que actúe, mediante un correo electrónico.

  Estos procesos de vigilancia fueron desarrollados (a modo de *scripts*) como parte del trabajo del proyecto MIRA y se describen en detalle en [Romeral 01]. En concreto:

  * mira_ps comprueba si los procesos se están ejecutando.
  * mira_ana gestiona el arranque, parada o reinicio de los procesos.
  * control.pl realiza la vigilancia, haciendo uso de los otros dos.

- Procesos del web dinámico. Los informes son presentados al consultor en forma de páginas web (ver apartado 2.3.3), de cuya generación se encarga un conjunto de procesos que parte de los ficheros VFL y que se integra con el propio servidor web (ajeno a la plataforma).

2.3.1.9 Consideraciones adicionales a los procesos

En primer lugar, en la cadena de proceso hay distintos puntos en los que se genera un único fichero de salida a partir de un conjunto de ficheros de entrada. En concreto *accum*, *biflow* y *mflow*. Es conveniente distinguir las diferencias existentes entre los tres casos:

- El proceso *accum* se encarga de *agregar* varios AFL en uno solo.

- El proceso *biflow* realiza la *correlación* de ficheros AFL de entrada y salida.
- El proceso *mflow* realiza la *consolidación* de ficheros VFL o REP correspondientes a distintos puntos de monitorización.

Si bien desde el punto de vista lingüístico podría considerarse que los términos agregar, correlar y consolidar son sinónimos, en este documento nos referiremos a ellos con el significado que se ha explicado anteriormente.

En segundo lugar, la descripción realizada en los apartados 2.3.1.1 a 2.3.1.7 tan solo considera el flujo de la información desde la captura hasta los informes finales, sin tener en cuenta que esta se produce en un marco temporal. Puesto que la información se captura y analiza en tiempo real y la información temporal interviene en la generación de los resultados (por ejemplo, gráficas diarias, medidas de tiempo de actividad de la plataforma, ...), vamos a analizar que mecanismos se incorporan en la plataforma para su control.

- El nombre de cada fichero contiene una marca de tiempo que indica en que momento se ha creado, con una resolución de un segundo. Así AFL010804173105 es un fichero de captura creado el 4 de Agosto de 2001, a las 17:31:05 horas. Para diferentes ficheros creados en el mismo segundo se usa una letra al final del nombre. Por ejemplo, los ficheros CAP020420100223a y CAP020420100223b fueron creados ambos el 20 de Abril de 2002, a las 10:02:23 horas.
- Adicionalmente, todos los ficheros incluyen internamente (la codificación exacta depende del tipo de fichero, ver formatos en apéndice A) marcas de instante inicial (T_0), instante final (T_f) y tiempo efectivo de captura(T_c).
  - T_0 indica el instante correspondiente al momento en el que se capturó el elemento de tráfico (paquete capturado, flujo, ..., depende del tipo de fichero) más antiguo del fichero. Se expresa en tiempo UNIX[15] (con *offset* de microsegundos).
  - T_f indica el instante correspondiente al momento en el que se capturó el elemento de tráfico (paquete capturado, flujo, ..., depende del tipo de fichero) más reciente del fichero. Se expresa en tiempo UNIX (con *offset* de microsegundos).
  - T_c se corresponde al tiempo real durante el que se estuvieron capturando datos y que no se corresponde necesariamente con la diferencia $T_f - T_0$. Se expresa como un intervalo en segundos y mi-

[15]Tiempo transcurrido desde las 00:00:00 GMT del 1 de Enero de 1970.

crosegundos. Observar que una medida de la eficiencia global de la plataforma es:

$$\eta(\%) = \frac{T_c}{T_f - T_0} \times 100$$

Observar que T_0 no es necesariamente consistente con la marca en el nombre del fichero. Si bien en un principio para los ficheros CAP puede ser así (o con un error muy pequeño), debido a los retardos de propagación de los datos por la cadena de proceso, el instante en el que cada proceso procesa los datos (y por tanto, momento de la creación del fichero y marca de tiempo en el nombre del mismo) será posterior al momento en el que se capturaron dichos datos.

A la hora de agregar, correlar o consolidar distintos ficheros de entrada, se realiza el adecuado tratamiento de estos valores. Sean $T_{01}, T_{02}, \ldots, T_{0N}$, $T_{f1}, T_{f2}, \ldots, T_{fN}$ y $T_{c1}, T_{c2}, \ldots, T_{cN}$ las marcas de los ficheros de entrada, las marcas del fichero de salida (T_{0S}, T_{fS}, T_{cS}) se calculan como:

$$T_{0S} = \min_{1 \leq i \leq N} T_{0i}$$

$$T_{fS} = \max_{1 \leq i \leq N} T_{fi}$$

$$T_{cS} = \sum_{i=1}^{N} T_{ci}$$

En el caso de ficheros BFL, VFL y REP (no consolidados) se consideran separadamente las marcas de sentido de entrada y sentido de salida. Para los ficheros consolidados (REP y SUMMARY) se separan, además del sentido entrada o salida, los m puntos de captura consolidados.

2.3.2 Infraestructura física

En el apartado 2.3.1 hemos hecho una descripción abstrayéndonos del soporte físico de los procesos de captura y análisis. Esto nos ha permitido centrarnos en el modelo funcional de la plataforma. No obstante, hemos de considerar también que existe una implementación física subyacente, importante porque va a condicionar el funcionamiento del sistema, y a cuya descripción dedicamos el presente apartado.

Uno de los requisitos de la plataforma es que fuera de bajo coste. Por tanto, se decidió utilizar PCs como equipos físicos. Para implementar los módulos funcionales, resulta muy útil la abstracción de *proceso concurrente* existente en los sistemas operativos UNIX (FreeBSD o GNU/Linux). La existencia de implementaciones libres[16] y gratuitas de dicho sistema operativo resulta muy conveniente desde el punto de vista del bajo coste.

A la hora de desarrollar los procesos, se eligió el lenguaje C/C++ para su implementación, pues proporciona el equilibrio adecuado entre bajo nivel (necesario para conseguir procesos eficientes) y alto nivel (programación estructura y orientada a objetos). No obstante, algunos procesos están desarrollados en otros lenguajes (procesos de vigilancia, procesos del web dinámico, ...), como Perl.

Si bien en un principio se podría pensar en realizar todos los procesos dentro de una única máquina conectada a la red monitorizada, lo que tendría sentido si esa máquina contara con recursos ilimitados, no es un enfoque óptimo para una arquitectura física basada en PCs. No todos los procesos de la cadena resultan igual de costosos en el consumo de recursos.

En concreto, los procesos que más capacidad de proceso de la CPU consumen (siendo este el recurso limitante sobre otros, como espacio de almacenamiento) son los de preprocesado. Es decir, *anaflow* constituye el cuello de botella de la cadena. Esto es debido a que este proceso tiene que hacer un gran número de búsquedas y comparaciones que algorítmicamente resultan costosas en la inspección del tráfico capturado.

Una configuración óptima es utilizar parejas de PCs.

- Máquina de captura. Es la máquina que esta conectada a la red monitorizada (mediante la correspondiente tarjeta de red) y que realiza el proceso *dumpcap*.

- Máquina analizadora[17]. Es la máquina que se encarga del preprocesado y análisis, con los procesos *anaflow*, *accum*, *biflow* y *vflow*.

La versatilidad de la orientación funcional a procesos es muy grande y podrían pensarse otras configuraciones. Una de ellas es utilizar tres máquinas en vez de dos, distinguiendo entre una máquina para *anaflow* y otra para el resto de procesos que siguen tras él, dejando toda la capacidad de proceso de un PC para el preprocesado. Sin embargo, esto incrementaría el coste en equipos un

[16]Brevemente, se entiende por software libre aquel cuyo código fuente puede ser abiertamente distribuido y modificado, bajo los términos de la licencia [GNU 89].

[17]Realmente, sería más apropiado el nombre "máquina preprocesador-analizador" pero usamos "máquina analizadora" para abreviar.

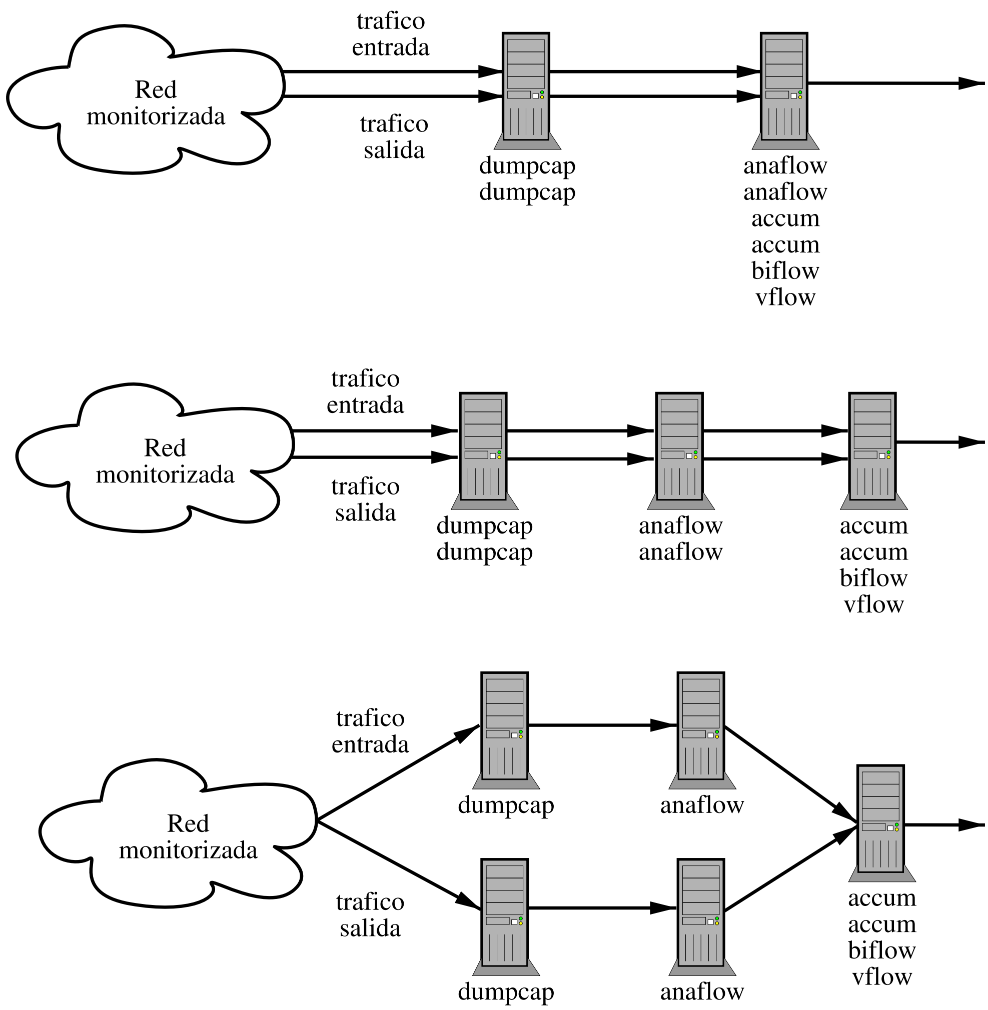


Figura 2.14: Alternativas de configuración física.

50% y no se ganaría mucho, pues los procesos que actúan tras *anaflow* trabajan con volúmenes tan pequeños (comparativamente) de datos que apenas consumen CPU. Otra posibilidad es utilizar cinco máquinas: dos de captura (cada una de ellas asociada a un sentido de tráfico), dos preprocesadoras asociadas a ellas corriendo únicamente *anaflow* y una quinta con *accum*, *biflow* y *vflow*. Sin embargo, aunque se ganaría en eficiencia, el coste que supone (150% más que la primera alternativa) hace que sea inviable. La figura 2.14 resume todas estas alternativas.

Por otra parte, es necesario realizar la consolidación de los resultados de todas las parejas de captura y análisis. Para realizar esa función se podría pensar en una máquina adicional de consolidación. No obstante, puesto que estos procesos tampoco suponen una elevada carga comparado con el preprocesado, en la práctica *mflow*, *sumrep* y los procesos del web dinámico se sitúan en la máquina analizadora de una de las parejas.

La comunicación entre procesos a través de ficheros se realiza con el uso de directorios, basándonos en el paradigma productor-consumidor. Un proceso proporciona sus ficheros de salida en un directorio predeterminado (establecido en el momento de arrancar el proceso como parte de su configuración). Este directorio será leído a su vez por el siguiente proceso de la cadena, en espera de nuevos ficheros de entrada que procesar. Una vez se ha procesado un fichero de entrada este se borra. Si no hay nuevos ficheros, el proceso queda en estado de suspenso durante un intervalo de tiempo, tras el cual vuelve a mirar el directorio. Observar que mediante este mecanismo un proceso que no tiene nada que hacer no está ocupando la CPU innecesariamente (puesto que estará en estado de suspenso), por lo que se garantiza que la capacidad de proceso es siempre la máxima posible para aquellos procesos que la necesitan realmente.

Para aquellos procesos que se encuentran en una misma máquina, no hay mayores problemas en la comunicación, gracias a la interfaz del sistema de ficheros que proporciona el sistema operativo nativo. Para procesos en distintas máquinas, se utiliza un sistema de ficheros distribuido (como NFS), que proporciona la misma abstracción pero haciendo uso de una red. Para asegurar la máxima transparencia en el uso de este servicio, las máquinas se comunican entre sí utilizando segmentos de Ethernet dedicados exclusivamente a ellas (es decir, no hay conectadas otras máquinas).

La presencia del cuello de botella en *anaflow* obliga a la utilización de mecanismos de control de flujo en *dumpcap*, ya que, puesto que la captura es más rápida que el preprocesado, se acumularían ficheros CAP hasta llegar a desbordar la capacidad de disco duro. Por ello, *dumpcap* fija un máximo de ficheros en su directorio de salida, sobrepasado el cual no sigue capturando hasta que haya disminuido el número.

En la práctica las máquinas de análisis suelen estar al 100% de carga de

CPU de proceso. Las máquinas de captura suelen estar mucho menos cargadas, con porcentajes del orden del 15%, precisamente debido a la espera obligada por el control de flujo[18].

Existe una limitación adicional a las tasas de análisis que podemos conseguir y es que, cuando los enlaces monitorizados son de alta capacidad (por ejemplo, un enlace ATM a 155 Mbps) las tarjetas de red tienen limitaciones para capturar todo el tráfico.

2.3.3 Acceso remoto a la plataforma

Es posible acceder a la plataforma de forma remota, utilizando Internet. Para ello, es necesario que las máquinas de captura tengan direcciones públicamente accesibles. Como ya mencionamos, existen dos tipos de agentes que interaccionan con la plataforma, cada uno de los cuales accede a la misma por distintos métodos.

- Administrador. Puesto que el administrador ha de realizar la configuración de los procesos de la plataforma para su correcto funcionamiento, es necesario que tenga acceso completo a las máquinas de las que consta la infraestructura. Por tanto, se utiliza un servidor de terminales remotos, slogin en concreto por las características de seguridad que incorpora (cifrado de la información en tránsito). También por razones de seguridad, solo se permite el acceso remoto a través de Internet a las máquinas de análisis: solo se puede acceder a las máquinas de captura desde la respectiva máquina de análisis de su pareja.

- Consultor. Los informes son presentados en forma de páginas web, a las que accederá el consultor por HTTP utilizando un navegador convencional (Netscape, Internet Explorer, . . .). Es necesario instalar un servidor web en la máquina de consolidación e integrarlo con los procesos que generan los informes de forma dinámica. Teniendo en cuenta que estamos utilizando sistemas operativos UNIX y software libre, la opción más aconsejable es Apache.

  Por otra parte, no debería permitirse que cualquiera pudiera acceder a los informes, puesto que contienen información sensible que solo deben conocer las personas adecuadas. La solución más simple para conseguirlo es utilizar una contraseña. Si se desea un nivel adicional de seguridad, puede considerarse el uso de certificados, utilizando el protocolo HTTPS.

[18] En sistemas operativos UNIX la herramienta *top* proporciona información sobre el uso de la CPU, tanto globalmente como para cada proceso.

2.4 Escenarios de aplicación

Existen distintos escenarios de aplicación en los que puede utilizarse la plataforma. Los más significativos, ya que engloban la mayoría de los casos, son la monitorización de una red troncal o la monitorización de una red local.

En todos los casos de aplicación la red monitorizada deberá estar basada en la pila de protocolos TCP/IP. Dado el abrumador dominio de TCP/IP como tecnología de red, este comentario carece casi de sentido en la actualidad.

- Monitorización de red troncal. En este escenario se realiza la monitorización del tráfico agregado que intercambian los sistemas administrativos que a dicha red troncal se conectan (estos sistemas administrativos serán las distintas subredes más elementales que la red troncal interconecta[19], ver figura 2.15). Puesto que este tipo de redes troncales están diseñadas para cursar gran cantidad de tráfico, utilizan enlaces de muy alta velocidad. Por tanto, debido a la capacidad de captura y análisis limitada, no podemos asumir que estemos capturando el 100% del tráfico cursado. Esto no es problema en la mayoría de los casos: el muestreo estadístico nos demuestra que basta con un porcentaje pequeño del capturas para extraer conclusiones válidas y útiles sobre el tráfico en su conjunto.

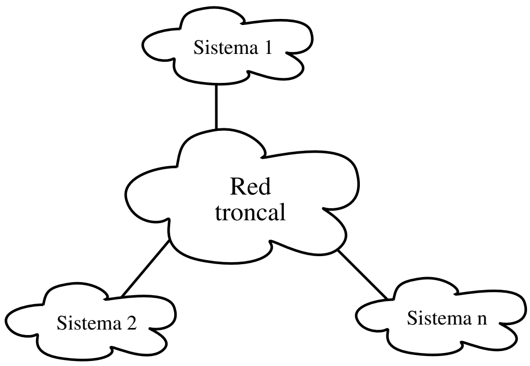


Figura 2.15: Red troncal clásica.

Para conseguir que la plataforma no sea intrusiva con la actividad normal de la red troncal hemos de situar una sonda o sensor que, a nivel físico, duplique el tráfico y lo dirija a la interfaz de captura. Por ejemplo, en el

[19]Por ejemplo, en el caso de RedIris, puede considerarse como sistemas las universidades, centros de investigación y demás organismos que a ella se conectan.

caso de una interfaz óptica, basta conectar un *spliter* (un acoplador 1 × 2) y al subsistema de captura de la plataforma[20].

- Monitorización de red local. La plataforma monitoriza es en este caso una red de área local. La diferencia significativa es que el tráfico que cursa una red de este tipo es mucho menor que el que cursa una red troncal, por lo que es posible (con el adecuado dimensionamiento de los recursos de captura y análisis) una captura total del tráfico cursado.

  Para que la plataforma no sea intrusiva con la actividad de la red local, aprovechamos las características difusivas de este tipo de redes[21] para duplicar cada paquete en todas las interfaces de escucha. Utilizando una interfaz Ethernet en modo promiscuo podemos capturar todo el tráfico que pase por la red.

  La posibilidad de utilizar la plataforma en estos escenarios, si bien plantea algunos problemas debido al diseño original de MEHARI (ver las notas a pie de página 6 y 9), también formaba parte del proyecto MIRA[22]. [Romeral 01] describe el desarrollo de un *dumpcap* para Ethernet.

2.4.1 Despliegue de MIRA en el troncal ATM de RedIris

La tecnología de red que se utiliza para la transmisión entre los nodos regionales y el nodo central donde se encuentra el router central es ATM. Pueden consultarse los fundamentos de esta tecnología en [McDysan+ 99], ya que aquí solo describiremos aquellos aspectos relacionados con la plataforma. A la hora de utilizar esta red para el envío de datagramas IP, surgen dos problemas:

- Utilización de un protocolo no orientado a conexión (IP) sobre otro que si lo es (ATM). Para solucionarlo se utiliza el método de provisión directa descrito en [ITU 93].

  Este método consiste en definir un CLS (*Connectionless Server*), de forma que cuando un nodo quiere enviar un datagrama IP por la red ATM

[20] Estrictamente, se produce una intrusión. La introducción de un elemento como es el acoplador en la fibra introduce pérdidas de inserción. Sin embargo, puesto que estas pérdidas solo afectan a la señal física, están en torno a 1 dB en el peor de los casos y el diseño de las interfaces ópticas de los equipos de interconexión se hace con márgenes de potencia amplios, no tiene el menor efecto.

[21] Ya sea porque el medio es realmente difusivo, como una Ethernet en bus o inalámbrica, o porque se consigue la difusión mediante elementos de repetición, como los *hubs* de interconexión.

[22] Dentro del paquete de trabajo "Consolidación de MEHARI" de la figura 2.2.

se lo envía a este servidor, que sabrá que hacer con él para encaminarlo hacia su destino. En RedIris, el papel del CLS lo toma el router central. En el nodo central existe un conmutador ATM a través del cual los nodos regionales tienen un circuito virtual permanente establecido con el router (identificado por un VPI/VCI). La capacidad de estos circuitos es de 155 Mbps y el tipo de servicio CBR (*Constant Bitrate*).

- Encapsulado de IP sobre ATM. La tecnología ATM se basa en la conmutación de células cuya carga útil es de 48 bytes. Para enviar datagramas IP es necesario realizar funciones de segmentación y reensamblado, de lo cual se encarga el nivel intermedio AAL5. Internamente, este nivel está dividido en dos: el subnivel CS y el subnivel SAR.

  Cuando tiene que enviarse un paquete del nivel superior, CS añade relleno y un *trailer* de 8 bytes[23]. De esta forma, la PDU tiene un tamaño total ajustado a un múltiplo de 48 bytes. Posteriormente, SAR segmenta esta PDU en fragmentos de 48 bytes, cada uno de los cuales se envía dentro de una célula. La última célula utiliza el bit de usuario de la cabecera del nivel ATM para indicar la finalización del paquete. En recepción, se realizan las funciones inversas para reconstruir el paquete. La figura 2.16 ilustra el proceso.

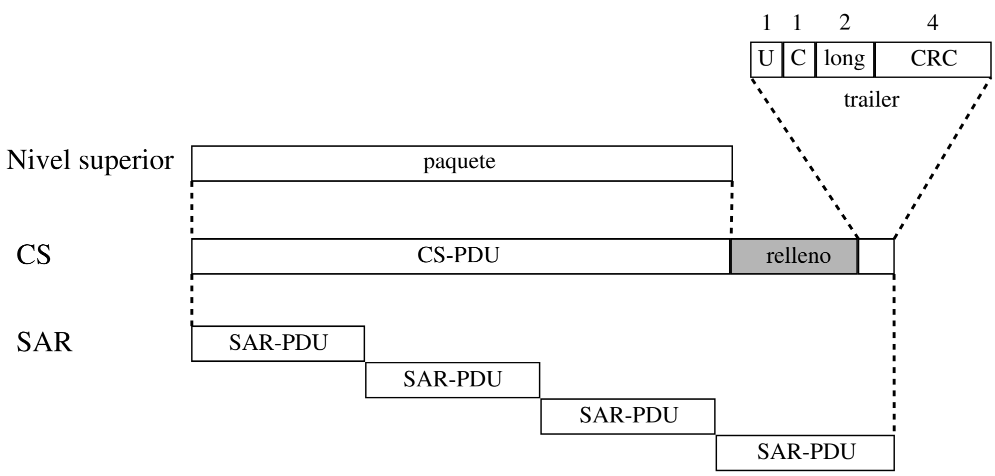


Figura 2.16: Nivel de adaptación AAL5.

[23]Los campos de este *trailer* son: UU (*User to User*), CPI (*Common Part Indicator*, no se utiliza), longitud (tamaño en bytes del paquete, sin incluir relleno ni *trailer*) y CRC (calculado sobre toda la trama, incluyendo paquete, relleno y *trailer*). No utilizaremos ninguno de ellos.

En [RFC 1483] se describen dos formas de encapsular protocolos sobre AAL5: con encapsulado LLC (para la transmisión de múltiples protocolos por un mismo circuito virtual, pero añadiendo una cabecera LLC/SNAP al paquete para diferenciar el protocolo) o multiplexación basada en circuito virtual (en la cual no se utiliza ninguna cabecera y los protocolos se distinguen porque cada uno utiliza un circuito virtual distinto). Puesto que en RedIris solo tenemos el protocolo IP, la alternativa más adecuada es esta última y es de hecho la situación en la cual se estaba utilizando la red ATM en el momento de abordar el desarrollo de la plataforma.

Para completar este análisis, señalamos que existe un estándar propuesto para la MTU de IP sobre ATM ([RFC 1577] y [RFC 1626]) que la sitúa en 9180 bytes. Datagramas de mayor tamaño deberán ser fragmentados antes de ser enviados por la red ATM, aunque esto no es habitual, debido a existencia de otras MTUs más restrictivas en las redes que puede haber atravesado el datagrama, como Ethernet (MTU 1500 bytes).

Cada uno de los nodos regionales está conectado con el router central mediante un circuito virtual permanente, identificado por un VPI/VCI. Estos circuitos pasan a través de un conmutador ATM, el cual está conectado al router central mediante seis pares de fibra óptica. Cada uno de estos pares consta de una fibra de salida (del nodo regional al router central) y otra de entrada (del router al nodo regional), con respecto a la Comunidad Autónoma. Las máquinas de captura tendrán dos tarjetas ATM, una de ellas para cada fibra del par.

Dada esta configuración de red, la plataforma se despliega con tres parejas de máquinas (capturadora y analizadora), tal y como muestra la figura 2.17. Cada una de las máquinas de captura dispondrá de dos tarjetas de red (una para analizar el tráfico de entrada del par y otra para el de salida), conectadas a las fibras mediante *spliters* ópticos. Las máquinas de captura se denominan cap0, cap1 y cap2. Las de análisis mira0, mira1 y mira2. Sus características técnicas se resumen en la tabla 2.1. Observar que debido a la existencia de seis pares de fibra pero solo tres parejas de máquinas, es necesario rotar las fibras a las cuales las máquinas están conectadas cada cierto tiempo si se desea tener medidas de tráfico de todas las Comunidades Autónomas.

En la práctica, no es posible registrar todo el tráfico, debido a las limitaciones en la capacidad de proceso ya comentadas. Las tasas de captura alcanzadas se encontraban en torno al 5% en enlaces ATM de 155 Mbps.

Las tres máquinas analizadoras se encuentran conectadas entre sí en una Ethernet dedicada, siendo una de las máquinas analizadoras (mira0 en concreto) la que realiza la consolidación y alberga el servidor web. La figura 2.18

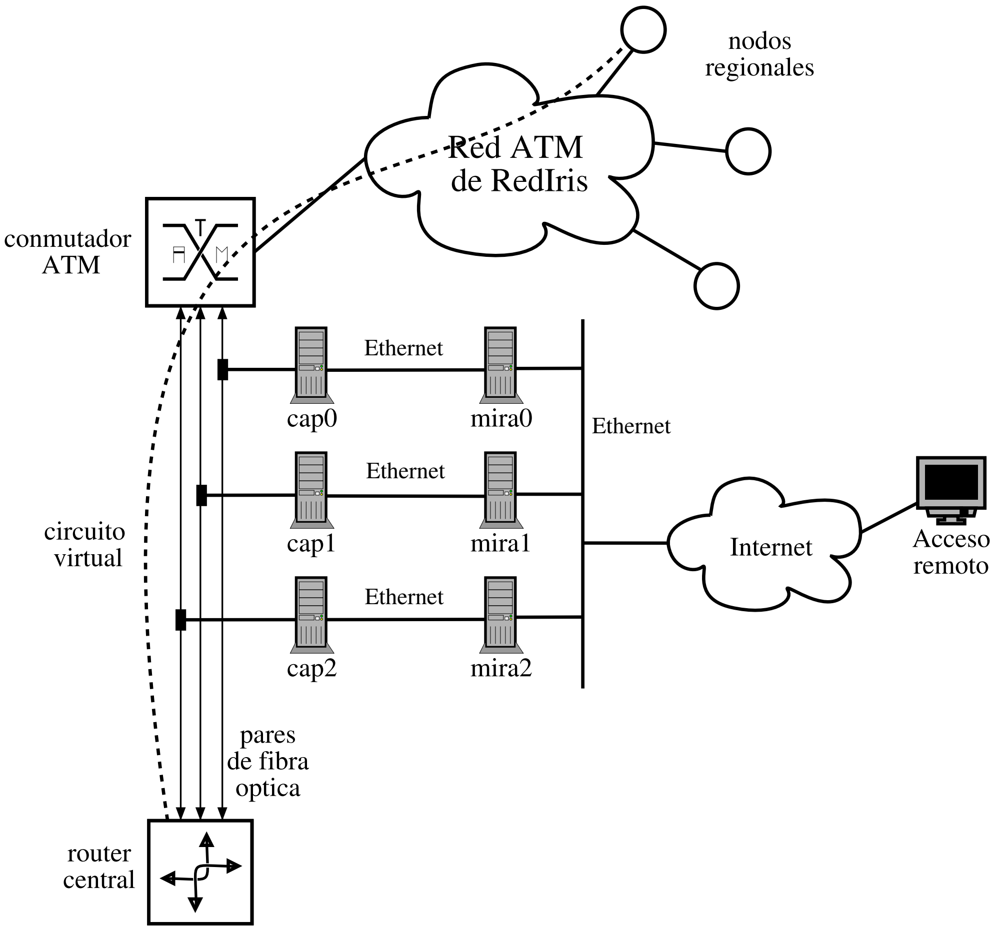


Figura 2.17: Escenario de despliegue de MIRA en RedIris.

<table>
<tr><td colspan="2">Capturadoras (cap0, cap1, cap2)</td></tr>
<tr><td>Hardware</td><td>Software</td></tr>
<tr><td>• PC con microprocesador Intel Pentium a 650 MHz
• 256 Mbytes de memoria RAM
• Disco duro IDE 8 Gbytes
• 1 Fast Ethernet 10 / 100Mbps
• 2 ATM NIC PCI SC-155, tipo Fore PCA-200EPC 256KB
• 2 divisores de fibra pasivos SC conectados a los puertos de Rx</td><td>• Sistema operativo FreeBSD 2.2.5
• OC3MON/BSD fatm-driver modificado por DIT-UPM
• OC3MON/BSD capture modificado por DIT-UPM
• Proceso dumpcap.
• Cliente NFS para el volcado de trazas hacia los procesos de la máquina analizadora.</td></tr>
<tr><td colspan="2">Analizadoras (mira0, mira1, mira2)</td></tr>
<tr><td>Hardware</td><td>Software</td></tr>
<tr><td>• PC con microprocesador Intel Pentium a 650 MHz
• 256 Mbytes de memoria RAM
• Disco duro SCSI 8 Gbytes
• 2 Fast Ethernet 10/100Mbps</td><td>• Sistema operativo Debian GNU/Linux 2.2.14
• Procesos propios de la plataforma MIRA.
• Servidor NFS para el volcado de trazas desde los procesos de la máquina capturadora.</td></tr>
</table>

Tabla 2.1: Características técnicas de los equipos en RedIris.

muestra la arquitectura completa de procesos[24]. Observar que los directorios cap, afl, afm y aaf acaban en 0 (para indicar tráfico de salida) o en 1 (para indicar tráfico de entrada).

Como ya se señaló, el proceso *dumpcap* es fuertemente dependiente del despliegue concreto de la plataforma y oculta detalles específicos de la conexión física a la red. Para el despliegue en el caso ATM podemos concretar como se produce la captura de tráfico en las máquinas capturadoras (figura 2.19).

- Los *drivers* de las tarjetas ATM leen las células que atraviesan los enlaces de fibra y las vuelcan en memoria en 16 bloques de 1 Mbyte.
- Una vez estos bloques están completos, el proceso *capture* se encarga de volcarlos a disco en ficheros BUF. No se realiza ningún tipo de procesamiento, por lo que estos ficheros son una reproducción exacta de lo que los drivers depositaron en memoria, con el único añadido de marcas de inicio, final y tiempo de captura.
- El proceso *dumpcap* toma como entrada los ficheros BUF, realizando la tarea de reensamblado de tramas AAL5 para obtener los datagramas IP, produciendo finalmente los ficheros CAP.

[24]Existe una diferencia con respecto a lo explicado en 2.3.1, ya que se usan dos procesos *accum* en vez de uno. El primero de ellos, realiza la acumulación minuto a minuto y el segundo hora a hora.

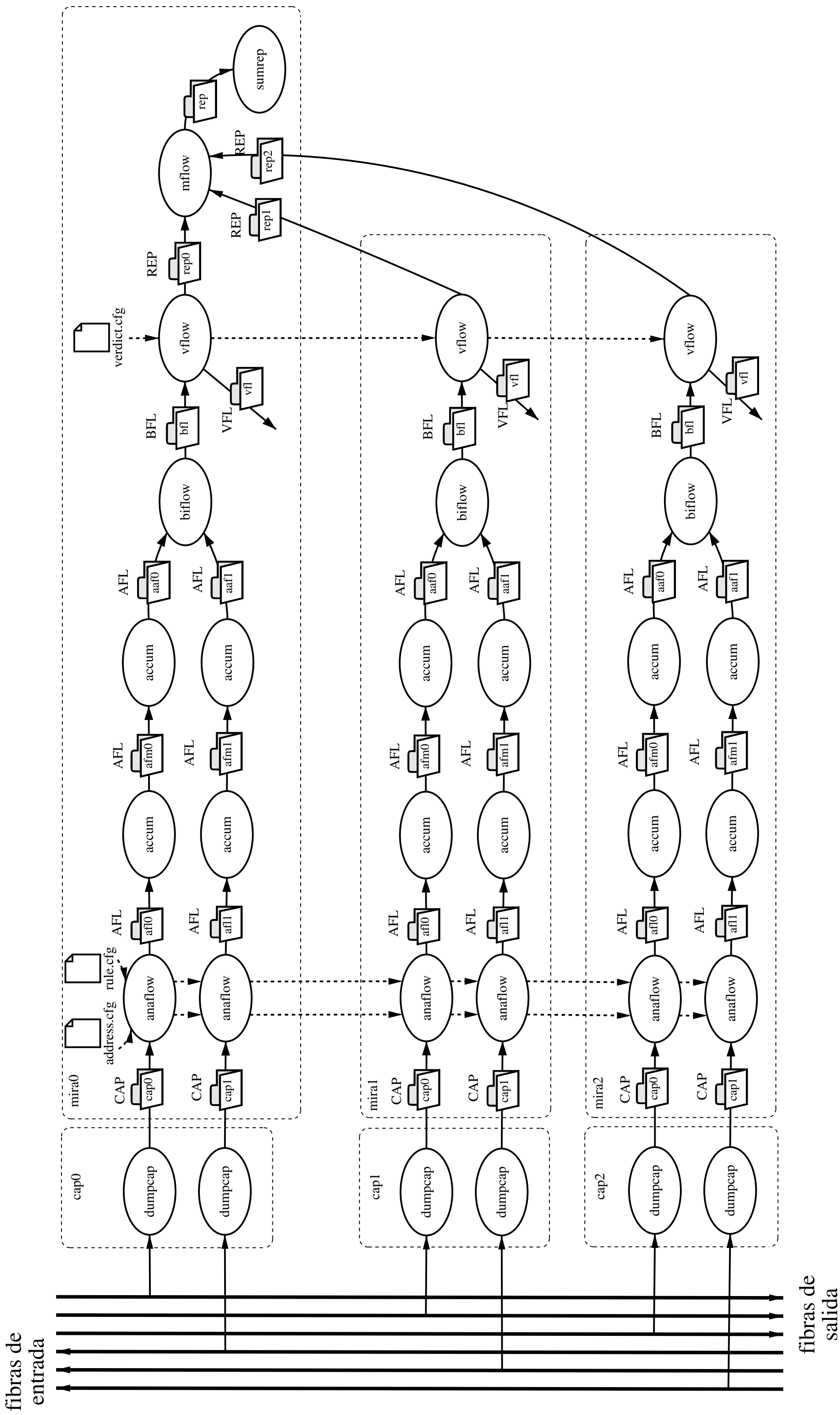


Figura 2.18: Arquitectura completa de procesos.

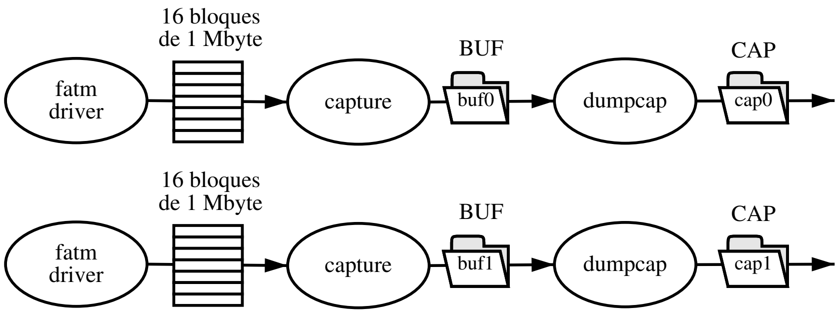


Figura 2.19: Captura de tráfico en redes ATM.

Capítulo 3

Detección de tráfico lúdico

3.1 Introducción

Dentro de la gran heterogeneidad de Internet, la popularización de esta red de comunicaciones ha alcanzado a un gran público que, cada vez más, la utiliza con fines lúdicos. Por ejemplo, consulta de resultados deportivos, carteleras de cine, páginas de grupos de música y muchas otras aplicaciones similares.

Dentro de estas actividades de tipo lúdico, destacan las de intercambio de contenidos entre usuarios. Si bien en un principio estos contenidos eran de tipo simple (textos o imágenes), en la actualidad los usuarios están más interesados en medios sofisticados, como audio y vídeo. Esta evolución de los intereses de los usuarios puede entenderse como una consecuencia del cambio de mentalidad en la concepción del PC, que de ser exclusivamente una herramienta de trabajo en sus orígenes se está convirtiendo en un centro de ocio multimedia.

No solo ha evolucionado el comportamiento de los usuarios, sino también las técnicas utilizadas para la realización de los intercambios. A las técnicas clásicas (como el FTP), se les han añadido un conjunto de protocolos y aplicaciones *peer to peer*, también conocido como P2P, cuya difusión y utilización ha sido creciente durante los últimos años. Con estos sistemas, son los propios usuarios los que intercambian los contenidos directamente, sin necesidad de servidores centrales que los almacenen. También hay que considerar los sistemas de *streaming*, donde los usuarios acceden a los contenidos en tiempo real.

No obstante, esta creciente utilización de Internet con fines lúdicos puede suponer problemas para los responsables de las redes, sobre todo cuando estas no están orientadas a la actividad lúdica[1]. En primer lugar, puede suponer un excesivo consumo de ancho de banda, que perjudique a los usuarios que

[1] Como es el caso de RedIris.

utilizan la red para los fines productivos para los que fue creada (académicos, corporativos, ...). En segundo lugar, muchas veces el intercambio de contenidos es una actividad, si no ilegal, al menos alegal, debido a que esos contenidos se encuentran protegidos por derechos de autor.

Por todo lo anterior, queda justificado el desarrollo e incorporación de técnicas especializadas en la detección de tráfico lúdico en la plataforma MIRA. De esta forma, el consultor será capaz de analizar el uso lúdico de su red y tomar las medidas oportunas al respecto, en función de la política de uso aceptable que se haya definido para la red monitorizada[2].

Puesto que abarcar todos los posibles contenidos lúdicos es una tarea inconmensurablemente compleja, nos hemos centrado en el audio MP3, debido a la gran popularidad con la que cuenta este formato en la actualidad. No obstante, en las conclusiones del apartado 3.6 se mencionan otro tipo de contenidos.

En primer lugar, realizaremos una tarea de análisis: estudio del formato MP3 (apartado 3.2) y protocolos de transmisión de audio (apartado 3.3). Posteriormente, nos centraremos en el diseño, analizando las distintas alternativas de mecanismos de detección de flujos MP3 que podemos utilizar, decidiendo justificadamente cual es la más conveniente (apartado 3.4). A continuación, se describirán las modificaciones en la plataforma que han sido implementadas junto con los resultados obtenidos (apartado 3.5), para finalizar con las conclusiones (apartado 3.6).

3.2 Estudio del audio MP3

El estándar MP3, desarrollado por el grupo MPEG de ISO y detallado en [ISO 11172] y [ISO 138138], es un algoritmo de codificación de fuente para audio. El objetivo que tiene es comprimir una fuente de audio lo máximo posible admitiendo ciertas pérdidas, pero sin que la calidad subjetiva que el oyente percibe se vea afectada apreciablemente. El factor de compresión que se alcanza depende de la fuente a comprimir, pero se pueden conseguir fácilmente relaciones 1:10 con una perdida de calidad inapreciable para un oyente humano.

Debido al alto factor de compresión que alcanza y a la existencia de gran número de codificadores y decodificadores software de gran calidad y libre distribución, en la actualidad MP3 se ha consolidado en Internet como el formato de distribución de audio por excelencia.

Es de destacar que el estándar MP3 describe tan solo la manera en la que

[2]Existe un debate abierto sobre la política de uso aceptable (permitir el tráfico, no permitir el tráfico lúdico, permitir el tráfico lúdico mientras no se perjudique la actividad normal de la red) que debe implantar una red académica y en el que nosotros no entraremos. En todo caso, son de gran interés los comentarios de [Echeverría 01].

ha de realizarse la codificación del audio, sin entrar en detalles de como ese audio va a ser almacenado o transportado a través de una red. Es por ello que en primer lugar estudiaremos los detalles de su estructura (en este apartado) para continuar con la descripción de protocolos y aplicaciones utilizadas para su transmisión (apartado 3.3).

3.2.1 Estructura del formato

Como ya se ha señalado, conceptualmente no existe diferencia entre un fichero almacenado en un disco duro o CDROM, un flujo de audio transmitido por Internet o los datos de audio en la memoria RAM de un reproductor portátil: todos ellos utilizan el mismo formato que aquí describiremos.

En realidad, ISO define dos tipos de audio: MPEG1 Audio (ISO/IEC 11172-3) y MPEG2 Audio (ISO/IEC 13818-3). También es posible una extensión de este último, denominada MPEG2.5 (estándar no oficial), para velocidades binarias muy bajas. Sobre esos tres formatos, se definen tres niveles de compresión (Capa I, II y III)[3], a cual más complejo y eficiente en la compresión.

Entendemos por *perfil* a la combinación de un tipo de audio y un nivel de compresión. Por tanto, van a existir 9 perfiles posibles, compartiendo ciertas características comunes en su formato.

El conjunto de bytes que conforman el audio lo denominamos *secuencia de audio* (figura 3.1). Observar que una secuencia de audio puede ser finita (por ejemplo, una canción almacenada en un fichero) o teóricamente infinita (por ejemplo, el audio emitido por una emisora radiofónica), un detalle que, como veremos más adelante, no es especialmente significativo[4].

Una secuencia de audio está formada por un conjunto de *tramas*. En Capa I y II, cada trama es independiente de las demás y como tal pueden ser decodificadas. En Capa III, debido a funciones de reserva que actúan como un buffer, una trama puede depender de las anteriores, es decir, se decodifican tramas conjuntamente. En el peor de los casos, el decodificador puede necesitar 9 tramas antes de poder decodificar la primera.

Esta característica permite decodificar las tramas en el momento en el que se reciben (o tras el pequeño *buffering* en el caso de Capa III), con lo que no es necesario conocer la secuencia de audio completa[5].

---

[3]Estrictamente hablando, el formato MP3 es el que usa la Capa III, aunque se ha popularizado el uso del término para toda la familia de estándares, y como tal lo estamos usando en este documento.

[4]En lo sucesivo, utilizaremos también el término fichero para referirnos a una secuencia de audio finita, y el término *stream* para referirnos a una secuencia de audio infinita.

[5]Por ejemplo, es posible empezar a reproducir un fichero MP3 que estamos descargando de Internet sin necesidad de que dicha descarga haya finalizado

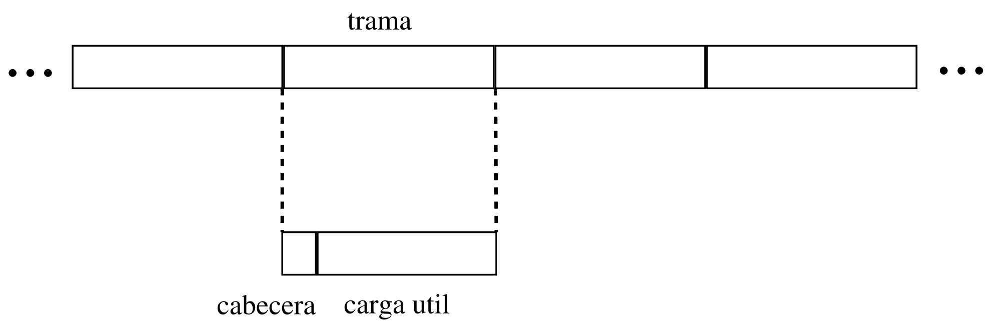


Figura 3.1: Secuencia de audio MP3.

Cada trama está formada por un número entero de intervalos (*slots*), cuyo tamaño es:

- Capa I: 4 bytes
- Capa II y III: 1 byte.

Obsérvese que en ambos casos, el intervalo está formado por un número entero de bytes, con lo que todas las tramas comparten la misma alineación a octeto, algo que, como veremos en el apartado 3.4, resulta muy conveniente a la hora de diseñar métodos de detección de flujos MP3.

Cada una de las tramas consta de una cabecera y una carga útil (*payload*), con la posibilidad de un CRC opcional entre ambos. La longitud de las distintas tramas que forman una secuencia de audio no es necesariamente homogénea.

Observar que no existe ninguna cabecera global ni similar que afecte a todo el conjunto de la secuencia de audio[6].

3.2.2 Componentes de trama

3.2.2.1 Cabecera

La cabecera tiene una longitud fija de 32 bits (4 bytes) y se sitúa al principio de la trama. Por tanto, ocupa el primer (en Capa I) o los cuatro primeros (en Capa II y III) intervalos de la trama. A continuación vamos a describir los distintos campos.

Palabra de sincronismo (Syn) 11 bits. Este campo tiene siempre el valor 1111 1111 111 y se usa para sincronizar la decodificación de audio, tal y como se describe en 3.2.3).

[6]Salvo que consideremos el ID3 Tag descrito en 3.2.2.4 como tal, aunque, como veremos, se trata de un campo poco relevante

<table>
<tr><td colspan="8">Syn</td></tr>
<tr><td colspan="3"></td><td colspan="2">ID</td><td colspan="2">Layer</td><td>CRC</td></tr>
<tr><td colspan="4">Bitrate</td><td colspan="2">Sample</td><td>P</td><td>V</td></tr>
<tr><td colspan="2">Mode</td><td colspan="2">Ext</td><td>C</td><td>O</td><td colspan="2">Emphasis</td></tr>
</table>

Figura 3.2: Formato de la cabecera MP3.

Identificador de formato MPEG (ID) 2 bits.

00	MPEG2.5
01	reservado
10	MPEG2
11	MPEG1

Identificador de nivel de compresión (Layer) 2 bits.

00	reservado
01	Capa III
10	Capa II
11	Capa I

Protección (CRC) 1 bit. Indica si se usa o no CRC (ver apartado 3.2.2.2).

Velocidad binaria (Bitrate) 4 bits. Indica la tasa binaria o *bitrate* de la fuente de audio comprimida. Cuanto mayor sea este bitrate, menor será la compresión del audio y mayor por tanto la calidad. La interpretación de este campo depende del ID y la Capa. La tabla 3.1 muestra los distintos valores posibles de bitrate (en Kbps) en función del código de este campo para los distintos casos de ID y Capa.

El bitrate libre (*free bitrate*) rara vez es utilizado. Puede encontrarse algún comentario adicional sobre el mismo en la nota a pie de página 11. El valor 1111 no está permitido.

Un bitrate muy utilizado es el de 128 Kbps. Observar que, si se compara con los 1.41 Mbps de velocidad binaria del audio sin comprimir de un CD Audio[7], la compresión MP3 supone una significativa reducción de aproximadamente 1:11.

[7] 2 canales $\times 16$ bits por muestra $\times 44100$ muestras por segundo $\simeq 1.41$ Mbps.

Valor	MPEG1 Capa I	MPEG1 Capa II	MPEG1 Capa III	MPEG2/2.5 Capa I	MPEG2/2.5 Capa II/III
0000	libre	libre	libre	libre	libre
0001	32	32	32	32	8
0010	64	48	40	48	16
0011	96	56	48	56	24
0100	128	64	56	64	32
0101	160	80	64	80	40
0110	192	96	80	96	48
0111	224	112	96	112	56
1000	256	128	112	128	64
1001	288	160	128	144	80
1010	320	192	160	160	96
1011	352	224	192	176	112
1100	384	256	224	192	128
1101	416	320	256	224	144
1110	448	384	320	256	160
1111	error	error	error	error	error

Tabla 3.1: Valores de bitrate.

Frecuencia de muestreo (Sample) 2 bits. Indica la frecuencia de muestreo del audio. Es necesario indicarla para el proceso de conversión de formato digital a analógico, previo a la reproducción del audio.

	MPEG1	MPEG2
00	44.1 kHz	22.05 kHz
01	48 kHz	24 kHz
10	32 kHz	16 kHz
11	reservado	reservado

MPEG2.5 utiliza frecuencias de muestreo que son la mitad que las que corresponden a MPEG2, si bien a efectos de cálculo de longitud de trama se utilizan las mismas (razón por la cual no se ha incluido en la tabla).

Obviamente, cuanto menor sea la frecuencia de muestreo, menor es el ancho de banda destinado al audio y por tanto, menor la calidad del mismo. El oído del oyente humano medio tiene una sensibilidad que va de los 20 Hz a los 20 kHz ([Menéndez 00]). Por tanto, según el criterio de Nyquist[8], la frecuencia de muestreo ha de ser como mínimo de 40 kHz para que el oyente no observe pérdida de calidad por reducción del ancho de banda.

Un valor muy adecuado resulta 44.1 kHz, que es la misma frecuencia de muestreo que utiliza el CD Audio.

Relleno (P) 1 bit. Indica si la trama incluye un intervalo extra de datos. Se utiliza cuando la frecuencia de muestreo es de 44.1 kHz (MPEG1), 22.05 kHz (MPEG2) o 11.025 kHz (MPEG2.5). Ver más adelante (apartado 3.2.3) que uso tiene.

Privado (V) 1 bit. Bit para uso privado. El estándar no define su uso, que será el que la aplicación de usuario quiera darle.

Modo de canal (Mode) 2 bits. Indica la forma de canalizar el audio: estéreo, estéreo combinado (*joint stereo*), dual o mono.

Extensión de modo (Ext) 2 bits. Cuando el modo es estéreo combinado, se utiliza este campo para indicar que subandas de audio se intensifican. No entraremos en más detalles sobre este campo.

Copyright (C) 1 bit. Si este bit vale 1, indica que el audio está protegido por derechos de autor, aunque el estándar no incorpora ningún mecanismo para asegurar que se protegen estos derechos.

Original (O) 1 bit. Si este bit vale 1, indica que el audio se encuentra en su medio original. En otro caso, es una copia.

Énfasis (Emphasis) 2 bits. Indica que tipo de filtros de de-énfasis debe usar el decodificador, cuando el audio ha sido sometido a un proceso de reducción de ruido al codificarlo[9]. Raramente se usa y no entraremos en más detalle sobre este campo.

3.2.2.2 CRC

Cuando el bit CRC de la cabecera vale cero, se inserta un código de redundancia cíclica de 16 bits (2 bytes). El CRC se realiza sobre los 16 últimos bits de la cabecera (es decir, quedan fuera los campos palabra de sincronismo, ID,

[8]Para poder reconstruir una señal analógica paso bajo de ancho de banda B ha de usarse una frecuencia de muestreo de al menos $f_s = 2B$.

[9]Esto es usado en algunos sistemas de codificación de audio, como Dolby.

Capa y CRC) y un fragmento de la carga útil (cuya longitud viene especificada en el estándar, así como los detalles acerca del polinomio generador del CRC).

En la práctica, el bit CRC suele valer 1, con lo que el CRC raramente se usa. Esto puede explicarse teniendo en cuenta que los errores que puedan producirse en una trama debidos a la codificación, almacenamiento o transmisión del audio van a ser muy infrecuentes, dada la tecnología actual (codificadores de gran calidad, medios de almacenamiento y protocolos de comunicación fiables). Puesto que la duración temporal del audio almacenado en una única trama es muy pequeña (como se explicará en 3.2.3), las ocasionales y escasas tramas corruptas no van a suponer una alteración significativa de la calidad del audio y, por tanto, no interesa complicar el codificador y decodificar con funciones de cálculo y comprobación del CRC.

3.2.2.3 Carga útil

La mayor parte de la trama almacena datos de audio comprimido. El algoritmo de compresión de audio que utiliza MP3 es de tipo predictivo con pérdidas, de la misma clase que el resto de estándares del grupo MPEG.

Los parámetros del algoritmo y la estructura del campo de carga útil de la trama están en función del tipo de audio (MPEG 1, 2 ó 2.5) y del nivel de compresión (Capa I, II ó III).

No entraremos en el detalle de como se codifica la fuente de audio para construir la carga útil de la trama, ya que el objetivo de nuestro trabajo es la identificación de flujos MP3 (en ningún caso la compresión y descompresión del audio) y, como se verá cuando analicemos los posibles métodos para llevarlo a cabo (apartado 3.4), no nos es necesario conocer en detalle la carga útil.

De esta forma, además, estamos eliminando la complejidad que supondría estudiar en detalle el algoritmo de compresión (una tarea no trivial).

3.2.2.4 ID3 Tag

Realmente, no se trata de un componente de trama, pero, puesto que aparece en muchos casos, lo describimos aquí. En cualquier caso, se trata de un componente completamente opcional.

Cuando la secuencia de audio es un fichero, es posible añadir a este un pequeño *trailer* al final del mismo[10] (figura 3.3). Este área de datos se denomina ID3 Tag y se utiliza para incluir información acerca del audio, como por ejemplo título, nombre del artista, album, año, género, etc. El ID3 Tag

[10] En una secuencia de audio infinita no tiene sentido, ya que no existe un “final” tras el cual podamos colocar este *trailer*.

empieza por el literal 'TAG' (en ASCII) y dentro se estructuran los distintos campos de información.

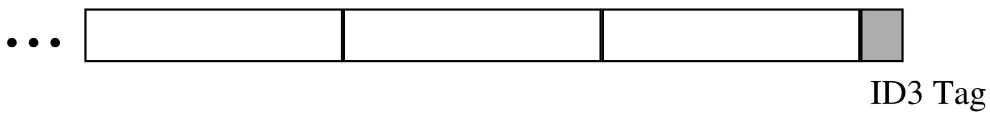


Figura 3.3: Posición del ID3 Tag en un fichero MP3.

El tamaño del ID3 Tag es de 128 bytes, que resulta despreciable frente a la longitud de un fichero típico MP3, teniendo en cuenta que esta suele encontrarse en torno a varios megabytes.

En realidad existen dos versiones de ID3 Tag, denominadas respectivamente ID3v1 e ID3v2. Lo anteriormente descrito se corresponde a la ID3v1. La ID3v2 es conceptualmente similar, pero se implementa de forma diferente. Se sitúa al principio de la secuencia de audio y puede contener contenidos complejos (por ejemplo, una imagen a modo de portada, las letras de una canción, ...). Su tamaño es mayor que en ID3v1, pero podemos considerarlo igualmente despreciable frente a la longitud total de un fichero MP3 típico.

Información sobre ambas versiones puede obtenerse en [ID3].

3.2.3 Sincronización y aspectos relacionados

A continuación, vamos a describir aspectos relacionados con la sincronización que realizan los reproductores de MP3 para decodificar una secuencia de audio, ya que en el estudio de su forma de funcionar podemos encontrar mecanismos útiles para la detección de flujos en la plataforma MIRA.

Ante un flujo de bytes donde se encuentra la secuencia de audio (fichero, audio en tiempo real, ...), lo primero que hace el decodificador es localizar una palabra de sincronismo. Es decir, se busca una secuencia de once bits con valor 1. De esta forma, se localiza la cabecera de la primera trama.

Para localizar las tramas subsiguientes, existe un procedimiento que permite calcular la longitud de una trama a partir de los datos de su cabecera. En concreto, con el bitrate, la frecuencia de muestreo y el bit de relleno (*padding*).

Para Capa I:

$$N = 12 \times \frac{R}{f}$$

Para Capa II y III:

$$N = 144 \times \frac{R}{f}$$

Donde N es la longitud de la trama (en intervalos), R es el bitrate (en bps) y f es la frecuencia de muestreo (en Hz). Si el tipo de audio es MPEG2.5, a pesar de que este modo define unas nuevas frecuencias de muestreo, han de usarse las de MPEG2 para el cálculo de N.

Para algunos casos de frecuencia de muestreo, puede ocurrir que N no resulte un valor entero y sea necesario redondear. Para ello se utiliza el bit de relleno: si vale 0, se rellena por defecto; si vale 1, por exceso. Precisamente la utilidad de este bit es ajustar el bitrate medio al tamaño necesariamente entero de las tramas[11].

Observar que para calcular el tamaño de una trama se usa información de la cabecera exclusivamente en esta misma trama. Por tanto, es posible tener una secuencia de audio donde el bitrate de cada trama sea distinto. Esta característica se denomina *variable bitrate* (VBR).

En realidad, el decodificador debería ser algo más cuidadoso, ya que es posible que al buscar la palabra de sincronismo, la encuentre no en una cabecera sino en otra parte de la trama (por ejemplo, en la carga útil). Por ello, es necesario verificar el sincronismo con un poco más de cuidado, siguiendo el proceso que se muestra en la figura 3.4.

Como puede verse, no basta con localizar la primera hipotética "cabecera" a partir de la primera palabra de sincronismo, sino comprobar que efectivamente la interpretación como tal de esos 4 bytes lleva a una posición de la secuencia de audio que es otra cabecera de trama. El parámetro C_{max} que se indica en el diagrama de flujo es el número de veces que debe repetir el decodificador esta comprobación. Cuanto mayor sea, más robusto será el proceso de sincronización. Sin embargo, con $C_{max} = 1$ es suficiente para garantizar que las probabilidades de sincronizarse erróneamente son prácticamente nulas.

En el caso de una secuencia de audio finita almacenada en un fichero podría pensarse en una manera trivial de realizar la sincronización: el primer byte del fichero es el principio de la primera trama. Sin embargo, es aconsejable seguir el procedimiento descrito que, si bien es más complejo, es más robusto, ya que en ocasiones (por fallos del compresor, presencia de un ID3v2 Tag, ...) los ficheros no comienzan con la secuencia de audio.

[11] Existe una variante de codificación en la que el bitrate no se especifica (*free bitrate*) y, puesto que no es posible calcular de esta manera el tamaño de trama, el proceso de sincronización de complica. Sin embargo, puesto que raramente se usa y además los decodificadores no están obligados a implementarla, no entraremos en ello.

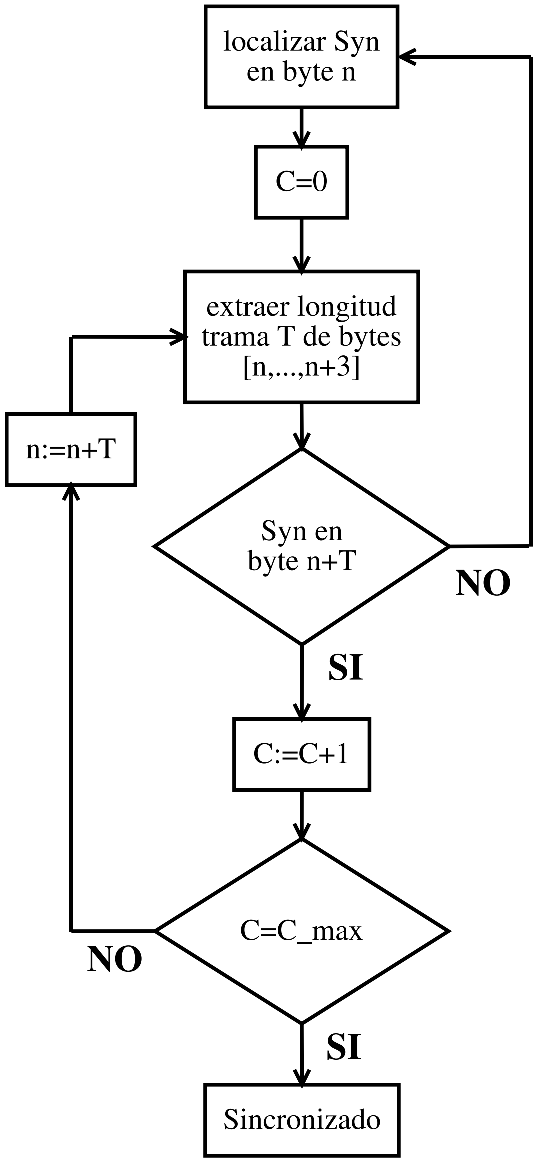


Figura 3.4: Proceso de sincronización.

Una vez el decodificador esté sincronizado con la secuencia de audio, puede extraer la carga útil de las tramas, decodificarlas aplicando en correspondiente algoritmo de descompresión y entregarlas al destino del audio (un fichero WAV, una tarjeta de sonido, ...).

Obsérvese que, a partir de la longitud de la secuencia de audio y del bitrate se puede calcular la duración del audio de forma aproximada (puede que no toda la longitud del fichero sea de audio, que la primera trama no coincida con el principio del fichero, que haya tramas con CRC erróneo que no se reproducen, ...).

$$T = 8 \times \frac{M}{R}$$

Donde T es la duración del audio (en segundos), M es el tamaño de la secuencia de audio (en bytes) y R es el bitrate (en bps).

3.2.4 Ejemplos

Para asentar los conceptos explicados en los anteriores apartados sobre las cabeceras y el cálculo del tamaño de trama, vamos a exponer tres ejemplos de secuencias de audio. Todos han sido extraídos de casos reales.

3.2.4.1 Audio a 48 kHz, 128 Kbps

<table>
<tr><td colspan="7">1111 1111 111</td></tr>
<tr><td colspan="2"></td><td colspan="2">11</td><td colspan="2">01</td><td>1</td></tr>
<tr><td colspan="3">1001</td><td colspan="2">01</td><td>0</td><td>0</td></tr>
<tr><td>01</td><td colspan="2">00</td><td>0</td><td>1</td><td colspan="2">00</td></tr>
</table>

Tabla 3.2: Cabecera del ejemplo FFFB 9444.

Consideremos un fichero cuyas tramas tienen cabecera FFFB 9444 (hexadecimal). La cabecera se muestra en la tabla 3.2. Analizamos los distintos campos.

- Palabra de sincronismo: 1111 1111 111.
- ID: 11. Es decir, MPEG1.
- Capa: 01. Por tanto, Capa III.

- CRC: 1. Sin CRC.
- Bitrate: 1001. Consultando la tabla 3.1, esto significa un bitrate de 128 Kbps.
- Frecuencia de muestreo: 01. Es decir, 48 kHz.
- Relleno: 0. Trama sin padding.
- Privado: 0.
- Modo: 01. Este valor indica estéreo combinado.
- Extensión de modo: 00. Modo del estéreo combinado.
- Copyright: 0. Sin derechos de autor.
- Original: 1.
- Énfasis: 00. Sin énfasis.

La longitud de cada trama se calcula con:

$$N = 144 \times \frac{128}{48} = 384$$

Es decir, 384 intervalos o (lo que es lo mismo, ya que en Capa III la longitud de intervalo es de 1 byte) 384 bytes. La duración del audio de una trama es de 24 ms. Observar que en este caso no es necesario hacer uso del bit de relleno.

3.2.4.2 Audio a 44.1 kHz, 128 Kbps

<table>
<tr><td colspan="7">1111 1111 111</td></tr>
<tr><td colspan="2"></td><td colspan="2">11</td><td colspan="2">01</td><td>1</td></tr>
<tr><td colspan="3">1001</td><td colspan="2">00</td><td>0</td><td>0</td></tr>
<tr><td>00</td><td colspan="2">00</td><td>0</td><td>1</td><td colspan="2">00</td></tr>
</table>

Tabla 3.3: Cabecera del ejemplo FFFB 9004.

Supongamos una cabecera FFFB 9004, que se muestra en la tabla 3.3. Identificamos los distintos campos:

- Palabra de sincronismo: 1111 1111 111.

- ID: 11. Es decir, MPEG1.
- Capa: 01. Por tanto, Capa III.
- CRC: 1. Sin CRC.
- Bitrate: 1001. Consultando la tabla 3.1, esto significa un bitrate de 128 Kbps.
- Frecuencia de muestreo: 00. Es decir, 44.1 kHz.
- Relleno: 0. Trama sin padding.
- Privado: 0.
- Modo: 00. Este valor indica estéreo.
- Extensión de modo: 00. Realmente, no sirve para nada, pues solo se usa cuando el modo es estéreo combinado.
- Copyright: 0. Sin derechos de autor.
- Original: 1.
- Énfasis: 00. Sin énfasis.

El cálculo de la longitud de la trama es el siguiente:

$$N = 144 \times \frac{128}{44.1} = 417.95$$

Hemos obtenido un valor no entero (ocurre cuando la frecuencia de muestreo es 44.1 kHz), por lo que es necesario redondear por defecto, puesto que el bit de relleno está a 0, a 417 bytes. Cada trama contiene información de 26.0625 ms de audio.

En la secuencia de audio nos encontraremos también con tramas con el bit de relleno activo. Es fácil comprobar que estas tendrán un cabecera FFFB 9204 y una longitud de 418 bytes. De hecho, si se estudia en detalle la estructura interna de la secuencia de audio, existe una secuencia regular de tramas, formada por 1 sin 417 bytes, seguida de 11 de 418 bytes, 1 de 417 y 36 de 418. Es decir, de cada 49 tramas, 47 son con relleno, lo que nos da un porcentaje del 95.92%. Esa secuencia de 49 tramas supone un tamaño de 20480 bytes, que se corresponden exactamente a 1.28 s.

<table>
<tr><td colspan="7">1111 1111 111</td></tr>
<tr><td colspan="2"></td><td colspan="2">00</td><td colspan="2">01</td><td>0</td></tr>
<tr><td colspan="3">0011</td><td colspan="2">00</td><td>0</td><td>0</td></tr>
<tr><td>01</td><td colspan="2">01</td><td>0</td><td>1</td><td colspan="2">00</td></tr>
</table>

Tabla 3.4: Cabecera del ejemplo FFE2 3054.

3.2.4.3 Audio a 11.025 kHz, 24 Kbps

Supongamos una cabecera FFE2 3054, que se muestra en la tabla 3.4. Identificamos los distintos campos:

- Palabra de sincronismo: 1111 1111 111.
- ID: 00. Es decir, MPEG2.5.
- Capa: 01. Por tanto, Capa III.
- CRC: 0. Con CRC.
- Bitrate: 0011. Consultando la tabla 3.1, esto significa un bitrate de 24 Kbps.
- Frecuencia de muestreo: 00. Es decir, 11.025 kHz, aunque para el cálculo de la longitud hemos de usar la que tendríamos con MPEG2, es decir 22.05 kHz.
- Relleno: 0. Trama sin padding.
- Privado: 0.
- Modo: 01. Este valor indica estéreo combinado.
- Extensión de modo: 01. Indica el modo de estéreo combinado.
- Copyright: 0. Sin derechos de autor.
- Original: 1.
- Énfasis: 00. Sin énfasis.

El cálculo de la longitud de la trama es el siguiente:

$$N = 144 \times \frac{24}{20.05} = 156.73$$

Que redondeado por defecto (relleno a 0) es una longitud de 156 bytes, equivalente a un audio de 52 ms exactos.

Es de destacar el uso del CRC en este caso. Es frecuente que las codificaciones con bajo bitrate sean utilizadas para su transmisión en tiempo real directamente por Internet (el *streaming* que describiremos en el apartado 3.3.2), por lo que en este caso el CRC si puede resultar útil.

3.3 Protocolos de transmisión de audio

En este apartado vamos a analizar los servicios, protocolos y aplicaciones que permiten la transmisión de datos MP3 a través de Internet, sin entrar en los detalles del formato MP3 (que ya han sido analizados en el apartado 3.2).

Podemos distinguir entre dos grandes grupos de aplicaciones como fuentes de los flujos MP3 que vamos a estar interesados en clasificar con la plataforma MIRA:

- Distribución de ficheros. Son aplicaciones (junto con el servicio y los protocolos asociados a las mismas) que establecen un diálogo con otra entidad (un servidor u otro usuario) para descargar un fichero MP3.

- *Streaming* de audio. Aplicaciones (junto con el servicio y los protocolos asociados) que permiten a un usuario o conjunto de usuarios escuchar una secuencia de audio MP3 en tiempo real.

La principal diferencia es que en el primer caso el fichero se descarga para ser reproducido después de la transmisión, mientras que en el segundo caso la secuencia de audio se decodifica y reproduce en tiempo real (análogamente a una emisora de radio convencional). En el primer caso, no existen requisitos temporales fuertes sobre la transmisión, mientras que en el segundo es necesario que los paquetes lleguen con el retardo adecuado al receptor. Las aplicaciones de *streaming* requieren de una calidad de servicio (QoS) para su disfrute por parte del usuario que Internet, hoy en día, no puede garantizar. No es de extrañar por tanto, que las aplicaciones de distribución de ficheros sean más populares y que las de *streaming* estén apareciendo en la actualidad de forma emergente.

3.3.1 Distribución de ficheros

Los primeros sistemas de distribución de ficheros eran particularizaciones de protocolos de propósito general ampliamente conocidos y utilizados. Así, la distribución de un fichero MP3 puede realizarse por descarga de un servidor mediante FTP o HTTP. Para este último protocolo está definido incluso el tipo MIME: audio/mpeg y aún hoy en día es ampliamente utilizado para la transmisión de MP3, ya que evita cortafuegos que puedan afectar a otros servicios más específicos (el puerto del HTTP suele estar abierto, ya que es el mismo protocolo se se utiliza para el acceso a la WWW).

En los sistemas anteriormente descritos, nos encontramos con una arquitectura cliente-servidor, en la que los usuarios se conectan a servidores donde se encuentran los ficheros que luego descargan. Sin embargo, recientemente ha surgido un nuevo paradigma de intercambio de ficheros, denominado *peer to peer* o P2P. En esta arquitectura, son los propios usuarios los que intercambian los ficheros entre ellos. También pueden existen servidores, pero su objetivo no es el de almacenar contenidos, sino el de poner en contacto a los clientes.

Aunque las primeras aplicaciones P2P surgieron para el intercambio de ficheros MP3, hoy en día su uso se ha extendido a todo tipo de contenidos que los usuarios quieran compartir (vídeo, imágenes, textos, ...). También incorporan muchas veces servicios añadidos, como la comunicación directa de usuarios mediante chat. Sin embargo, nosotros nos centraremos exclusivamente en MP3.

Esta nueva filosofía tiene algunas implicaciones importantes. La primera de ellas es que el valor de una aplicación de este tipo ya no radica en la cantidad y calidad de los contenidos de los servidores de quien proporciona el servicio, sino en los propios usuarios. En este sentido, aplicando el concepto de externalidad de red, si n es el número de usuarios de un servicio basado en interconexión (como es el caso del P2P) la externalidad (valor o utilidad de ese servicio) es proporcional a n^2. También hay que tener en cuenta la "calidad" de estos usuarios: ancho de banda con el que ofrecen sus contenidos (por ejemplo, en función de si se conectan con un módem telefónico o con ADSL), que contenidos está dispuesto a compartir cada uno (existen usuarios "egoístas", muy interesados en descargar cosas de los demás, pero poco en ofrecer las propias), etc.

En segundo lugar, puesto que el proveedor no mantiene servidores de ficheros, no es responsable (al menos directamente) de los contenidos que intercambian los usuarios. Los servidores tan solo actúan como intermediarios (*brokers*). Esta cuestión es importante, ya que muchas veces los contenidos están protegidos por derechos de autor y su intercambio es ilegal. De esta forma, la responsabilidad legal recae sobre los usuarios, no sobre el servidor[12].

En las aplicaciones P2P existen realmente dos protocolos, uno de ellos entre cliente y servidor para acciones de registro, búsqueda, etc. y otro cliente-cliente para el intercambio de ficheros entre usuarios.

Existen dos tipos de aplicaciones P2P, en función de como se realice la funcionalidad del servidor. En los sistemas *centralizados*, los servidores son máquinas separadas de los clientes, que mantienen una visión global del conjunto de usuarios conectados y los ficheros que comparte cada uno. En los sistemas *descentralizados*, los propios clientes actúan también como servidores y las búsquedas de ficheros se propagan de unos a otros recursivamente mediante un mecanismo de "cuchicheo".

El procedimiento para un sistema centralizado es en líneas generales es el que se muestra en la figura 3.5.

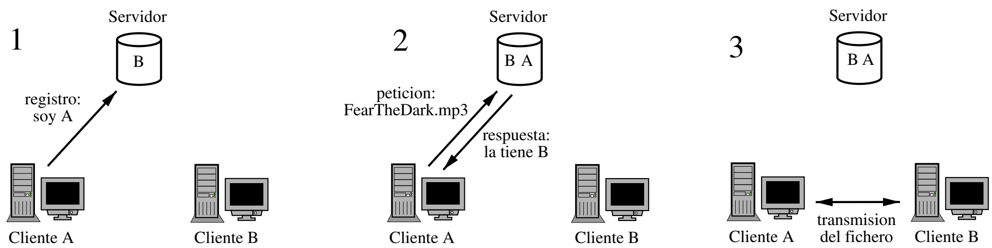


Figura 3.5: P2P centralizado.

1. Cuando un cliente quiere conectarse al servicio, lo primero que hace es registrarse en un servidor[13], enviándole en este registro una lista de los ficheros que comparte ese usuario. El servidor mantiene en todo momento una lista de usuarios conectados, junto con que contenidos ofrece cada uno.

2. Una vez conectados, un cliente puede realizar búsquedas de contenidos. Por ejemplo, una canción identificada por un título, un artista, una determinada calidad mínima, etc. Para ello, realiza una petición al servidor el cual consulta su lista de usuarios y le devuelve una lista de resultados,

[12]Realmente, este es un asunto polémico, ya que no está claro hasta que punto es responsable el proveedor del servicio de facilitar intercambios ilegales entre sus usuarios. En el pasado, empresas como Napster han sufrido acciones legales debido a problemas de este tipo ([ELMUNDO 00]).

[13]Hablamos de un solo servidor por simplificar, aunque, por razones de escalabilidad lo que realmente existe es una red de servidores conectados entre sí.

entre los que se encuentran las direcciones de los usuarios que tienen los contenidos de interés.

3. El cliente establece una conexión con el otro cliente y descarga el fichero correspondiente.

El caso de un sistema descentralizado se muestra en la figura 3.6.

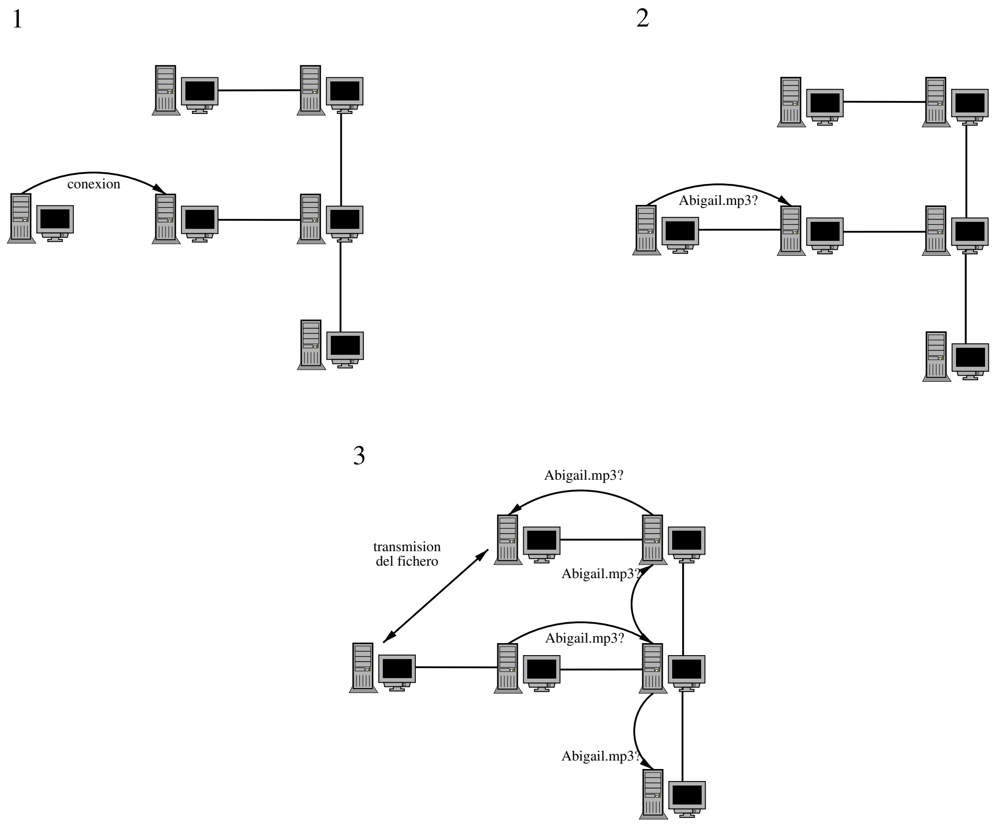


Figura 3.6: P2P descentralizado.

1. El cliente establece una conexión con otro cliente (o varios), cuya dirección debe conocer de antemano.

2. Para realizar una búsqueda de fichero, el cliente envía una petición indicando lo que está buscando a todos aquellos otros clientes con los que está conectado.

3. Los clientes retransmiten la petición recursivamente. La profundidad de esta recursión (es decir, el número de veces que puede retransmitirse) puede controlarse estableciendo un tiempo de vida en la petición inicial. Una vez encontrado algún nodo que posee el fichero buscado, este se transmite al cliente original directamente.

A continuación vamos a describir con más detalle algunas aplicaciones P2P. Los protocolos pueden ser propietarios o públicamente conocidos, permitiendo este último caso implementaciones tanto privadas como propietarias de los clientes y servidores. En concreto, nos centraremos en el sistema Napster, que, aunque es relativamente antiguo, es todavía muy utilizado en la actualidad.

3.3.1.1 Napster

El protocolo Napster fue definido como primer sistema P2P de amplia difusión por la empresa homónima. El protocolo en sí es abierto, aunque las implementaciones de clientes y servidores no tienen por que serlo[14]. A continuación describiremos la forma en la que el usuario se comunica con los servidores.

El protocolo es de tipo centralizado. Se utiliza TCP como nivel de transporte. Los mensajes intercambiados contienen los siguientes campos:

Longitud 2 bytes. Indica la longitud del área de datos.

Tipo 2 bytes. Indica el tipo de mensaje.

Datos Una cadena de caracteres en ASCII plano. Su formato dependerá de tipo de mensaje y constará de una serie de campos, separados entre ellos por un blanco (ASCII 32)[15].

En este estudio no se explicarán los detalles del protocolo relativo a tipos y formato de los mensajes (información detallada puede encontrase en [Napster]), pero veremos, en líneas generales, el procedimiento que se sigue desde que un usuario se conecta al servidor hasta que empieza a transferir un fichero (figura 3.7).

1. En primer lugar, el cliente A tiene que encontrar un servidor de la red Napster. Para ello, se realiza una consulta a un *metaservidor* (por ejemplo, server.napster.com), que corre típicamente en el puerto 8875 o 8876.

[14]Entre las implementaciones propietarias se encuentran las de la propia Napster, o el cliente WinMX. Entre las implementaciones abiertas, destacamos el servidor OpenNap.

[15]Si un campo incluye espacios se entrecomilla para evitar confusiones. Por ejemplo, “nap v0.8”.

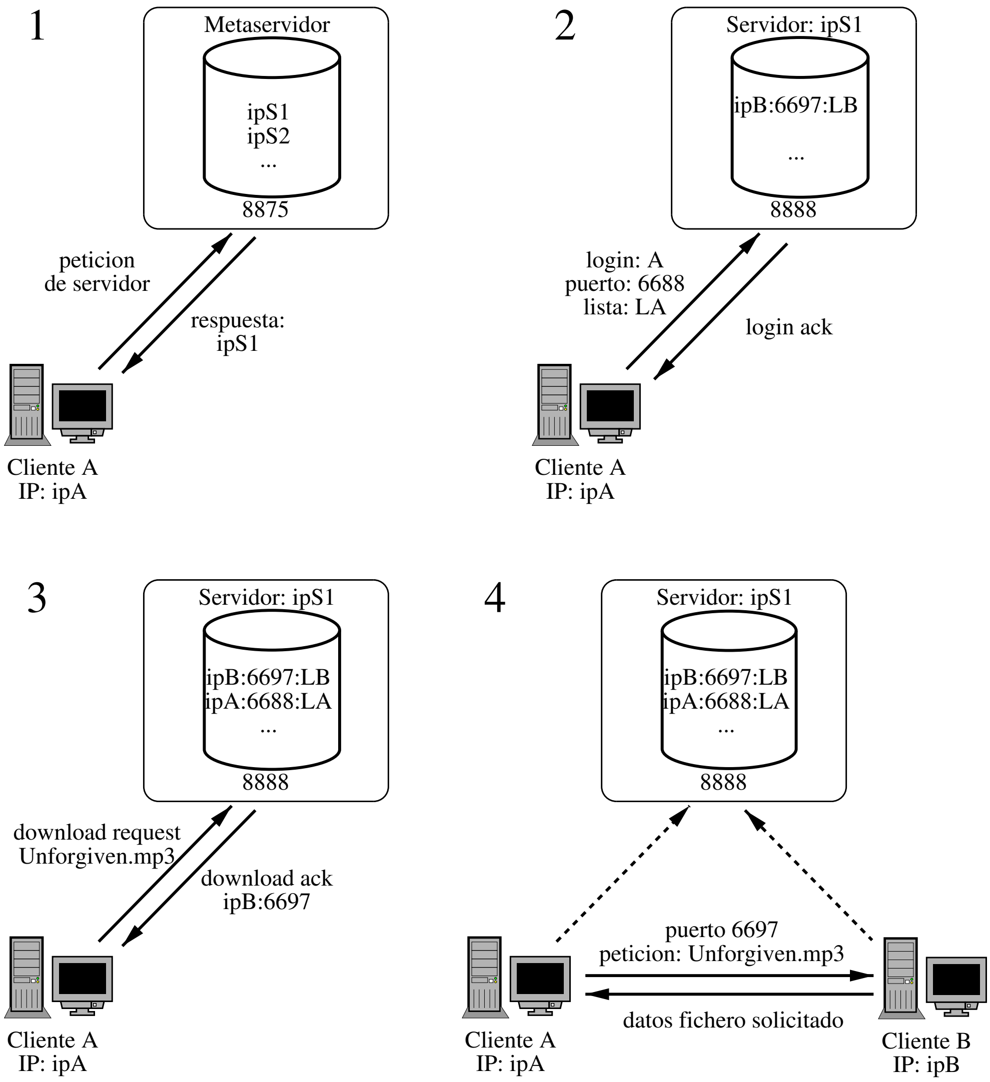


Figura 3.7: Protocolo Napster.

2. Una vez encontrado un servidor, el cliente A se registra en él (los servidores escuchan en los puertos 8888 y 7777 típicamente, aunque también pueden usarse los puertos 4444, 5555 y 6666), mediante su login y su password. En este proceso, se especifica el puerto de datos que utilizará el cliente A (6688, 6697, 6699 o 4329, típicamente; también puede especificar 0, indicando de esta forma que se encuentra detrás de un cortafuegos que le impide aceptar conexiones entrantes) para las transferencias.

3. El cliente A consultará los ficheros, realizará búsquedas, etc. hasta que encuentre el fichero MP3 deseado, momento en el que envía al servidor un mensaje de *download request*. El servidor responde con un *download ack* en el que, entre otras cosas, se encuentra la dirección IP y puerto de datos del cliente B que tiene el MP3 deseado.

4. El cliente A abre una conexión con el cliente B en el puerto especificado, entablando una comunicación cliente-cliente que da lugar a la transferencia del fichero. Los dos clientes informan al servidor de en que estado se encuentran: avisan cuando han empezado la transferencia y cuando ha terminado. De esta forma, el servidor puede gestionar la conexión entre ambos y actuar intercambiando mensajes de error.

Existe un procedimiento alternativo si el cliente B está detrás de un cortafuegos que le impide recibir conexiones entrantes. El este caso, cuando el cliente A recibe el *download ack* con un valor de puerto 0, envía un *alternative download request* al servidor. El servidor envía entonces un *alternative download ack* al cliente B (con la dirección IP y el puerto de datos del cliente A) y es B quien abre la conexión cliente-cliente[16].

3.3.1.2 Otros sistemas P2P

A continuación se mencionan otros protocolos y aplicaciones, sin entrar en tantos detalles con respecto al análisis que hemos hecho de Napster. Puede obtenerse información útil sobre sistemas P2P en [infoAn] y [Evans 01].

El sistema AudioGalaxy es muy similar a Napster, salvo que los clientes utilizan un puerto estándar para comunicarse con el servidor, lo que permite evitar cortafuegos específicos. Para ello se utiliza el puerto TCP 21, asociado al control de FTP. Cuando se busca un fichero, el servidor comunica la dirección y el puerto (en principio aleatoriamente distribuido entre el 41000 y el 50000) de otro usuario que pueda transferirlo.

[16] Obviamente, si ambos clientes A y B se encuentran tras cortafuegos que impiden conexiones entrantes, no es posible el intercambio de ningún modo.

Gnutella y KaZaA son sistemas distribuidos, en los que los clientes actúan como servidores. Se utilizan los puertos 6346 y 6347 para Gnutella y 1214 para KaZaA. Su principal problema (presente en general en todo sistema distribuido) es conectarse a la red de usuarios, ya que en principio las direcciones de estos no tienen por que ser públicamente conocidas, como es el caso de los servidores centralizados.

Lo habitual es que los programas cliente utilicen un solo sistema P2P, existiendo una identificación clara con el mismo. Así, hablamos de clientes para Napster, clientes para KaZaA, etc. Sin embargo, están empezando a aparecer clientes multiprotocolo, en los que el usuario especifica que es lo que quiere buscar y el cliente se encarga de rastrear distintos sistemas para encontrarlo. Un ejemplo de esto es MP3Rage, que integra Napster y Gnutella.

3.3.2 *Streaming*

Estos sistemas constan típicamente de un servidor que reproduce una secuencia de audio y un conjunto de clientes que se conectan al mismo para recibirla y reproducirla en tiempo real. Realmente, el *streaming* es un concepto general, que puede aplicarse no solo a audio, sino a cualquier contenido multimedia susceptible de reproducción en tiempo real (como por ejemplo el vídeo).

Normalmente, el audio que se reproduce es una secuencia infinita, no está asociado a un fichero. En este sentido, estos servidores se comportan como emisoras de radio convencional, salvo que en vez de usar el espectro electromagnético utilizan Internet como medio de difusión.

No obstante, hay que tener en cuenta una importante diferencia con respecto a la radiodifusión clásica, y es que Internet no es un medio difusivo. En un principio, podría pensarse en el uso de direcciones de *multicast* para las aplicaciones de *streaming*. Sin embargo, el uso de *multicast* a gran escala es todavía materia de investigación en Internet. Por ello, lo que se utiliza es un servidor que envía la secuencia de audio a cada uno de los clientes, junto con un mecanismo que permita a un cliente negociar la conexión con el servidor para que este le envíe el flujo.

Internet no garantiza la necesaria calidad de servicio (es decir, no garantiza un ancho de banda ni requisitos de retardo en la transmisión de audio). La transmisión del audio desde el servidor a los clientes se hace "lo mejor posible" (*best-effort*) y, para evitar problemas, se suelen utilizar codificaciones MP3 de baja calidad (y por tanto, con pocos requisitos de ancho de banda) en comparación a la que se utiliza en los ficheros de las aplicaciones de intercambio. El ejemplo del apartado 3.2.4.3 ha sido extraído del flujo de una aplicación de *streaming*.

Como protocolo de transporte, lo más adecuado es utilizar UDP. No es un

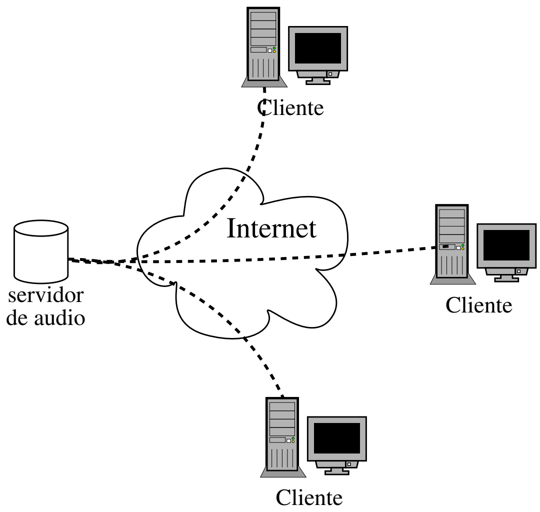


Figura 3.8: *Streaming* de audio por Internet.

protocolo fiable y pueden perderse algunas tramas de audio, pero se pueden asumir pérdidas en la reproducción del audio. TCP, aunque es fiable, no da buenos resultados cuando se utiliza para la transmisión de datos en tiempo real. De todas formas, algunas aplicaciones (como Shoutcast, que describiremos inmediatamente) utilizan TCP, porque les permite realizar la petición de audio y la recepción de la secuencia simultáneamente[17].

Como ejemplo de aplicación de *streaming* de MP3 podemos mencionar [Shoutcast], que es un protocolo muy sencillo en el cual el cliente (por ejemplo, un reproductor Winamp) abre una conexión con el servidor de audio y realiza un GET de HTTP. El servidor responde con la cabecera típica (HTTP/1.0 200 OK), tras lo cual empieza a mandar la secuencia de audio por tramas. Los servidores tienen asignados una serie de puertos, que son 8000, 8001, 8600, 8700, 8800 y 8888[18], pero en la práctica se puede utilizar cualquiera (como ejemplo, una de las emisoras usada en las pruebas corría en el puerto 13704).

3.4 Mecanismos de detección de flujos

Una vez realizada la exposición de los aspectos más relevantes tanto del formato MP3 (apartado 3.2) como de los protocolos y aplicaciones utilizadas para

[17]Esto es, el mismo *socket* que abre el cliente al servidor para realizar la petición de audio es el que utiliza el servidor para enviar de vuelta la secuencia de audio.

[18]Observar que el puerto 8888 también es usado en Napster.

su transmisión (apartado 3.3) vamos a considerar que mecanismos podemos incorporar a la plataforma MIRA para la detección de este tipo de tráfico.

Las técnicas que podemos emplear pueden clasificarse en dos grupos. Por una parte, aquellas que hacen uso de la inspección de paquetes y por otra aquellas que se basan en direcciones y puertos. Vamos a analizar cada una de ellas.

3.4.1 Inspección de paquetes

La determinación de si un paquete pertenece a un flujo MP3 puede hacerse mediante la inspección de su contenido, localizando en el mismo la presencia de determinados patrones. Vamos a considerar detenidamente que patrones son los más adecuados.

En primer lugar, hemos de descartar la posibilidad de basarnos en el área de carga útil de la trama, ya que el contenido de la misma es completamente aleatorio y no se localiza en ella ningún patrón con más frecuencia que otro. Esto puede explicarse desde el punto de vista de la teoría de la información, teniendo en cuenta que la carga útil contiene datos de audio comprimidos, donde no puede haber patrones que sean más frecuentes que otros. Si así fuera, la compresión no sería eficiente. Igualmente, tampoco nos podemos basar en el CRC, que es un campo que ni siquiera es obligatorio.

Por tanto, hemos de basar los patrones en la cabecera de las tramas. Teniendo en cuenta el análisis realizado en 3.2.2.1, la posibilidad más inmediata parece la detección de la palabra de sincronismo, ya que esta identifica claramente el comienzo de una trama. Sin embargo, un patrón consistente tan solo en once bits con valor 1 es demasiado corto y resultará demasiado frecuente en todo tipo de flujos como para resultar útil. Si únicamente usáramos eso, estaríamos detectando erróneamente un gran número de flujos que no son MP3. No resulta, por tanto, un heurístico adecuado.

Para evitar estas equivocaciones, podríamos reproducir el proceso que sigue un decodificador al sincronizarse con un flujo (explicado en 3.2.3). Es decir, comprobar la presencia de dos tramas consecutivas, basándose para ello en el cálculo de longitud a través de la supuesta primera cabecera.

A pesar de que este mecanismo de detección sería bastante robusto, plantea varios problemas:

- Supone una complicación significativa de la plataforma. En primer lugar, se produciría un incremento en la carga de trabajo, debido a que hay que realizar la decodificación de la cabecera para calcular su longitud y la comprobación de la segunda cabecera. Pero lo más costoso sería modificar el proceso *anaflow*. Hasta ahora dicho proceso trabajaba detectando patrones dentro de los flujos, sin necesidad de mantener un estado de los mismos (apartado 2.3.1.2). Si incorporamos este mecanismo, habría que señalar de alguna manera cuando detectamos una palabra de sincronismo que se está a la espera de comprobar de nuevo el patrón en una posición determinada.

- No tenemos el flujo completo. El mecanismo de sincronización funciona bien cuando el decodificador recibe toda la secuencia de audio. Sin embargo, en la plataforma MIRA existen escenarios en los cuales, debido a la diferencia entre la velocidad de transmisión de la red monitorizada y la velocidad de proceso (menor la segunda que la primera) no se capturan todos los paquetes de un mismo flujo (normalmente, esto ocurre en los escenarios de red troncal, descritos en 2.4). Por tanto, la comprobación de dos tramas consecutivas puede ser imposible si la primera se encuentra en un paquete y la segunda en otro y este último no es capturado. Este problema se acentúa debido a que los datagramas IP pueden fragmentarse.

- No se detectarían correctamente flujos MP3 multiplexados. En ocasiones, el audio MP3 se encuentra multiplexado dentro de un flujo multimedia (por ejemplo, vídeo y audio). Sería interesante que los mecanismos incorporados a la plataforma pudieran categorizar estos flujos como tráfico lúdico por su presencia de MP3. Sin embargo, el cálculo de posición de la segunda cabecera en el mecanismo de sincronización supone que la secuencia de audio ya está demultiplexada. Incluir métodos de cálculo de está posición en función del método de multiplexación o la demultiplexación del flujos multimedia resultaría una tarea excesivamente compleja.

Por todo lo anterior, descartamos este mecanismo de sincronización como método útil para la detección de flujos MP3.

Una alternativa más fácil de realizar sería la detección simple de patrones, extendiendo la palabra de sincronismo con los campos de ID y Capa. De esta forma, evitamos el problema de frecuentes detecciones erróneas que tendríamos de utilizar tan solo la palabra de sincronismo. Sin embargo, estamos perdiendo generalidad, ya que mientras que la palabra de sincronismo está presente en la cabecera de las tramas de todos los perfiles de codificación, con ID y Capa estamos restringiendo a casos concretos.

No obstante, en la práctica no se usan todos los perfiles habitualmente, siendo el MPEG1 Capa III el más extendido en la actualidad. Recientemente, con la aparición de las aplicaciones de *streaming* se están empezando a popularizar también el MPEG2.5 Capa III.

- MPEG1 Capa III. Patrón: 1111 1111 1111 101.
- MPEG2.5 Capa III. Patrón: 1111 1111 1110 001.

Como puede verse, ambos patrones constan de 15 bits. Podría pensarse todavía que es un patrón demasiado corto y considerar su extensión con los campos de bitrate y frecuencia de muestreo, basándonos de nuevo en que existen ciertos subformatos que en la práctica son los que más se usan[19].

- MPEG1 Capa III. 128/160/192 Kbps, 44.1 kHz (típico de distribución de ficheros).
- MPEG2.5 Capa III. 24 Kbps, 11.025 kHz (típico de aplicaciones de *streaming*).

De esta forma, podemos extender el patrón a 21 bits (tabla 3.5).

MPEG	Patrón (binario)	Patrón (hexadecimal)
Bitrate 128, frec. 44.1	1111 1111 1111 101* 1001 00**	FFF{A,B}9{0,1,2,3}
Bitrate 160, frec. 44.1	1111 1111 1111 101* 1010 00**	FFF{A,B}A{0,1,2,3}
Bitrate 192, frec. 44.1	1111 1111 1111 101* 1011 00**	FFF{A,B}B{0,1,2,3}
Bitrate 24, frec. 11.025	1111 1111 1110 001* 0011 00**	FFE{2,3}3{0,1,2,3}

Tabla 3.5: Patrones utilizados para la detección MP3.

Una posibilidad adicional haciendo uso de la inspección de paquetes sería la búsqueda de patrones claves de los protocolos de distribución o *streaming*. Por ejemplo, el patrón 'audio/mpeg', que identifica el tipo MIME asociado a MP3, o palabras claves del protocolo Napster ('nap v', '.mp3', …).

Por último, destacar que las técnicas que hemos descrito basadas en la detección de patrones funcionarán mal cuando el flujo se transcodifica de algún modo. Si bien estos patrones siguen funcionando en un flujo multiplexado (inconveniente que como vimos tenía la técnica basada en comprobar la sincronización), fallarán si al flujo se le aplica algún filtro para comprimir o encriptar. Aplicar un filtro para comprimir (por ejemplo, gzip) no tendría mucho sentido, pues MP3 es un formato de compresión muy eficiente y poco se puede ganar al aplicar un segundo compresor. Sin embargo, la encriptación de flujos es posible en algunas aplicaciones de distribución de ficheros, como [Filetopia].

[19] Evidencias experimentales de ello pueden encontrarse en el apartado 3.5.2.1.

3.4.2 Direcciones y puertos

Haciendo uso de las cabeceras de IP y transporte de los paquetes y sin necesidad de inspeccionar su contenido, podemos intentar clasificar un flujo como MP3. Todas estas soluciones se basan en el conocimiento de los protocolos de distribución de ficheros y *streaming*.

En primer lugar, aunque se use FTP o HTTP para la descarga de ficheros MP3, no podemos basarnos en los puertos que utilizan (20 y 21 para FTP, 80 para HTTP). Estos protocolos son demasiado genéricos y se utilizan para la transmisión de todo tipo de datos, con lo que no se puede clasificar MP3 (ni ninguna otra cosa) en función de ellos.

Por tanto, hemos de considerar protocolos más específicos. Las aplicaciones de distribución de ficheros y *streaming* utilizan en la mayor parte de las ocasiones unos puertos predefinidos y que no suelen utilizar otras aplicaciones. Por ejemplo, los puertos 1214, 4329, 4444, 5555, 6346, 6347, 6666, 6688, 6697, 6699, 7777, 8875, 8876 y 8888 son utilizados por Napster y otros sistemas P2P, o los puertos 8000, 8001, 8600, 8700, 8800 y 8888, utilizados por Shoutcast[20].

Otra alternativa de clasificación puede hacerse en base a una lista de direcciones IP asociadas a redes de intercambio[21] o *streaming*. No se trata de incluir todos los servidores que las forman (esto podría mermar el rendimiento de MIRA, al tener que verificar gran número de direcciones), pero sí la de los servidores más conocidos o estables (por ejemplo, aquellos que actúan de metaservidores, como search.napster.com, o de emisoras).

Existen dos inconvenientes a este tipo de técnicas. En primer lugar, que son demasiado dependientes de la aplicación concreta de distribución de ficheros: si el proveedor de servicio cambia la dirección de sus servidores o si se cambia el protocolo para que se usen otros puertos distintos, los heurísticos de clasificación fallarán. En segundo lugar, en gran parte de los casos no puede identificarse una dirección IP ni un puerto de transporte. Esto que es típico en la mayor parte de las aplicaciones P2P, donde los clientes tienen una dirección IP aleatoria y la transmisión se produce haciendo uso de puertos aleatorios, también se da en algunas aplicaciones de *streaming*[22].

3.4.3 Evaluación de alternativas

Una vez analizadas las distintas posibilidades, detallaremos cuales son las modificaciones finalmente incorporadas a la plataforma MIRA.

[20] Como se explicó, respectivamente, en los apartados 3.3.1 y 3.3.2.

[21] Una lista de servidores Napster puede encontrarse en [Napigator].

[22] En concreto, servidores de Shoutcast que no utilizan los puertos típicos.

En primer lugar, de entre los dos grupos de técnicas de detección de MP3 descritos anteriormente (inspección de paquetes y uso de direcciones y puertos), consideramos que las que más posibilidades ofrecen son las de inspección de paquetes. Existen tres razones principales para ello.

- El uso de direcciones y puertos está demasiado ligado a unos protocolos concretos. Esto no sería mayor problema si no fuera porque el escenario MP3 está demasiado ligado a factores ajenos al mismo (modas, cierres judiciales, ...) y la utilización de estos protocolos es demasiado inestable y cambiante con el tiempo[23].
- Como ya se comentó en 3.4.2, gran parte del tráfico MP3 se produce entre direcciones IP y puertos aleatorios.
- La filosofía de MIRA confía siempre que se pueda en la inspección de contenidos, ya que es este uno de los principales aspectos que diferencian la plataforma de aquellas otras que se basan tan solo en las cabeceras IP y de transporte. Recuérdese que el módulo de clasificación anticipada basada en direcciones y puertos se incorporó para mejorar la eficiencia reduciendo la carga de trabajo de *anaflow* (ver apartado 2.3.1.5).

Por tanto, utilizaremos técnicas basadas en la inspección de paquetes para la búsqueda de patrones característicos de flujos MP3.

Concretamente, la implementación ha consistido en lo siguiente:

1. En primer lugar, verificar que los patrones determinados en 3.4.1 son realmente eficaces. Para ello, se estudió un conjunto de datos, tanto MP3 como no MP3, comprobando con que frecuencia aparecen en ambos casos.
2. Una vez comprobado cuales son los patrones útiles, se modificó la configuración de la plataforma para incluirlos en el análisis de síntomas y en la clasificación en categorías de tráfico.
3. Por último, se analizaron los resultados obtenidos y se refinó la configuración de la plataforma para hacerla más eficiente.

La realización de estas tareas se encuentra detallada en el apartado 3.5.

El que nos hayamos decido por técnicas de inspección de paquetes no quiere decir que la clasificación en función de direcciones y puertos quede totalmente

[23]Esta razón descarta también el uso de inspección de paquetes para encontrar palabras claves de protocolos.

desestimada, ya que siempre puede usarse en combinación con las anteriores para reforzar los mecanismos de clasificación. Por ejemplo, podemos incluir las reglas de la figura 3.6 dentro del fichero de clasificación de direcciones address.cfg[24].

64.124.41.19|*|LUD|server.napster.com
0.0.0.0/0|6688|LUD|Napster
0.0.0.0/0|6697|LUD|Napster
0.0.0.0/0|6699|LUD|Napster
0.0.0.0/0|4329|LUD|Napster
0.0.0.0/0|4444|LUD|Napster
0.0.0.0/0|8000|LUD|Shoutcast
0.0.0.0/0|8001|LUD|Shoutcast
etc.

Tabla 3.6: Reglas de detección MP3 basadas en direcciones y puertos.

3.5 Implementación y resultados

3.5.1 Herramientas para el trabajo con MP3

Para realizar el estudio del formato MP3 ha sido necesaria la utilización de una serie de herramientas. Afortunadamente, gracias a la gran popularidad de este formato de compresión, existen gran número de herramientas de gran calidad que pueden ser utilizadas, muchas de ellas de libre uso y distribución.

Para la generación de MP3 a partir de ficheros de audio no comprimido (WAV, CD Audio, ...) pueden utilizarse herramientas de compresión, algunas de las cuales permiten fijar las opciones de codificación (bitrate, frecuencia de muestreo, ...) con bastante flexibilidad. Por ejemplo, la herramienta [AudioCatalyst].

Para el estudio de ficheros MP3 existentes, podemos utilizar herramientas como [XMMS] (X Windows) o [Winamp] (Microsoft Windows) que, si bien pensadas principalmente como reproductores, también permiten obtener información acerca de los parámetros del audio comprimido.

[24]Observar que, para aquellos casos en los que se clasifique en función a un puerto podríamos utilizar como alternativa la configuración como puertos especiales en la configuración de *vflow*. Sin embargo, puesto que el fichero de clasificación de direcciones se evalúa antes de que *vflow* realice la decisión de veredicto, la solución aquí señalada sería más eficiente.

No obstante, la información proporcionada por herramientas del tipo de las anteriores es muchas veces imprecisa o inexacta, por lo que se ha desarrollado una herramienta para el estudio de secuencias de audio MP3. Dicha herramienta ha sido profusamente utilizada en la realización del estudio estadístico del apartado 3.5.2.

La herramienta se denomina *mp3h* y ha sido desarrollado en C para GNU/Linux. Es parecida a la herramienta [mp3_check], de similares características, pero que no se utilizó finalmente porque no era lo suficientemente flexible. Se invoca desde la consola de comandos y admite dos argumentos: fichero MP3 de entrada que va a ser analizado y fichero de salida donde se realizará un volcado de las cabeceras.

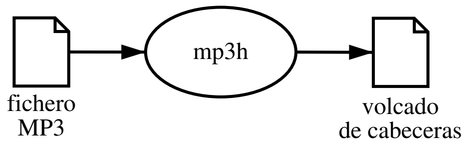


Figura 3.9: Herramienta *mp3h*.

El programa busca la primera cabecera de trama mediante el proceso descrito en el apartado 3.2.3, tras lo cual recorre el fichero procesando cada trama hasta alcanzar el final del mismo (bien porque se llegó efectivamente al final del fichero, bien porque se alcanzó un ID3 Tag, bien porque se encontró una cabecera errónea).

El procesado de una trama consiste simplemente en el volcado (hexadecimal) de su cabecera a un fichero de salida, junto con la posición dentro del fichero donde se ha encontrado. Posteriormente, puede examinarse ese fichero para realizar algún tipo de postprocesado (por ejemplo, calcular la longitud de las tramas, calcular porcentaje de tramas con el bit de relleno activo, ...).

El programa también muestra informaciones útiles por la consola, como una representación de la primera cabecera que encuentra, número de tramas totales encontradas, ...

3.5.2 Estudio estadístico

El objetivo que nos planteamos en este apartado es comprobar si los patrones introducidos anteriormente resultan útiles para su uso dentro de la plataforma MIRA. En concreto, vamos a analizar la frecuencia de aparición de los patrones FFFA y FFFB (correspondiente a MPEG1 Capa III) para el caso de ficheros MP3 y los patrones FFE2 y FFE3 (correspondiente a MPEG2.5 Capa III) para el caso de flujos de *streaming* en un conjunto de datos, que constituirá el conjunto de prueba.

Observar que estos son los patrones de 15 bits señalados en el apartado 3.4.1 extendidos a 16 bits (2 bytes), con los posibles valores del bit CRC. Por simplicidad, no consideramos en este análisis los patrones más largos (21 bits), que son los que definitivamente se configuraron en la plataforma y serán objeto de un análisis más detallado en la evaluación de resultados (apartado 3.5.4).

Las pruebas realizadas han consistido en la búsqueda de estos patrones en los datos del conjunto de prueba. Sin embargo, es importante señalar que el número de patrones encontrados en un determinado fichero o flujo (ocurrencias) no es importante en términos absolutos, sino en relación con el tamaño del fichero o flujo. Por ejemplo, que un fichero de 10 bytes tenga 5 ocurrencias FFFA es mucho más significativo que un fichero de 10 Mbytes tenga 100 ocurrencias.

Por ello, para cada fichero se calcula el siguiente ratio, que es lo que se usa para comparar los resultados, donde L es la longitud del patrón (en bytes), N es el número de ocurrencias y T es la longitud del fichero o flujo (en bytes).

$$R(\%) = L \times \frac{N}{T} \times 100$$

En nuestro caso la longitud de los patrones es de 2 bytes. Observar que un ratio de 100% significa que tenemos un fichero formado por una sucesión ininterrumpida de patrones (por ejemplo, FFFA FFFA FFFA ...)[25].

3.5.2.1 Ficheros MP3

El conjunto de prueba utilizado consiste en unos 30.000 ficheros (con un volumen total de 17 Gbytes de información) tanto MP3 como no MP3 elegidos de forma aleatoria[26]. Los patrones analizados son FFFA y FFFB.

Las conclusiones obtenidas son las siguientes:

- El patrón FFFB es más significativo que el FFFA en los ficheros MP3. Como ya se comentó en 3.2.2.2, el software de compresión MP3 rara vez utiliza CRC.

- Considerando este patrón (FFFB), los ratios que resultan de analizar los MP3 del conjunto de prueba son los que se muestran a continuación.

[25] Puede comprobarse fácilmente si consideramos $T = NL$ en la ecuación anterior.

[26] En el caso MP3 se han elegido ficheros con canciones extraídas de colecciones de música. En el caso no MP3 se han elegido ficheros aleatoriamente escogidos del sistema operativo.

$R \approx 0.50$ % (en la mayor parte de los ficheros)
$R \approx 0.40$ %
$R \approx 0.33$ %
$R \approx 0.57$ % (en escasos ficheros)
$R \approx 0.68$ % (en escasos ficheros)

La aparición de estos resultados tiene su explicación teórica. Supuesto que el fichero MP3 solo presenta el patrón en su cabecera (es decir, no lo presenta dentro de la carga útil), es fácil calcular el ratio teórico esperado. Por ejemplo, para una codificación de 128 Kbps a 44.1 kHz (esto es, longitud media de 417.95 bytes, como ya se vio en 3.2.4.2) y supuesto un fichero de n tramas, el ratio esperado es:

$$R(\%) = 2 \times \frac{n}{417.95 \times n} \times 100 = 0.478$$

Podemos construir la tabla 3.7, donde vemos asociados los valores teóricos con los esperados y comprobamos que ambos valores se aproximan bastante.

Codificación	Ratio teórico(%)	Ratio experimental(%)
192 Kbps	0.319	≈ 0.33
160 Kbps	0.382	≈ 0.40
128 Kbps	0.478	≈ 0.50
112 Kbps	0.547	≈ 0.57
96 Kbps	0.638	≈ 0.68

Tabla 3.7: Estudio estadístico de ficheros MP3.

El ligero incremento que se produce en el ratio experimental con respecto al teórico (de 0.05% en el mayor de los casos) se debe a que nuestra suposición inicial acerca de la no aparición del patrón en la carga útil no es del todo realista. Dichas ocurrencias elevan ligeramente los valores del ratio esperado.

El hecho de que el resultado de 0.5% sea el que aparece en la mayor parte de los ficheros, es que la codificación más utilizada es la de 128 Kbps. Las codificaciones de 112 y 96 Kbps y por debajo son muy poco utilizadas, y por tanto, apenas aparecen ficheros esos ratios.

- Los patrones FFFB y FFFA resultan muy poco frecuentes en el conjunto de ficheros no MP3. Salvo casos muy excepcionales, el ratio de ocurrencias apenas sobrepasa los 0.05% como máximo (lo normal es que ni siguiera aparezca el patrón, o que no sobrepase el 0.01%).

- No se ha detectado ningún formato de fichero (a parte de MP3, como es obvio) en el que estos patrones resulten excepcionalmente frecuentes. La única excepción son los formatos BMP e ICO (*bitmaps*), cuyos ficheros en ocasiones presentan frecuentes ocurrencias FFFB. Sin embargo, esto no debe preocuparnos, ya que pocos ficheros en estos formatos se transmiten por Internet y, además, al ser imágenes podemos asociarlos fácilmente al tráfico lúdico, con lo que aunque nos equivoquemos al detectar en ellos síntomas MP3, no se comete un gran error. De todos modos, es de esperar que el uso de patrones largos (21 bits) evite gran parte de estas falsas detecciones.

3.5.2.2 Flujo de *streaming*

Se trata de realizar el mismo estudio que en el apartado anterior, pero utilizando los patrones de búsqueda FFE2 y FFE3. Sin embargo, como consecuencia del poco auge de las aplicaciones de *streaming* en la actualidad, no resulta tan fácil encontrar un conjunto de prueba grande como en el caso de los ficheros.

Finalmente, el conjunto de prueba consistió en el flujo capturado de una emisora Shoutcast de Internet. Para obtener dicho flujo, se utilizó la herramienta [Ethereal], que permite fácilmente capturar un flujo TCP y grabarlo a un fichero, para su posterior análisis.

No repetiremos la exposición de resultados anteriores, puesto que las conclusiones son muy similares. Al igual que para los ficheros, el CRC parece ser poco utilizado[27] por lo que el patrón que suele aparecer es el FFE3 y el ratio de aparición en el flujo MP3 es alto cuando se lo compara con el de los ficheros no MP3 (y también puede comprobarse como se acerca a lo esperado teóricamente, $R = 1.276$ %).

3.5.3 Modificación en la plataforma

Hemos de modificar dos aspectos de la configuración de la plataforma.

- Configuración de *anaflow* (rule.cfg). Añadir al conjunto de reglas de análisis las que permitan la detección de un nuevo síntoma, asociado al MP3.

[27] Aunque en este caso, puede deberse a que Shoutcast usa como nivel de transporte TCP, que ya garantiza la fiabilidad. Posiblemente si se utilizaría CRC en el caso de que el transporte fuera con UDP.

- Configuración de *vflow* (verdict.cfg). Modificar las reglas de clasificación para asociar el síntoma MP3 a la categoría de tráfico lúdico (LUD).

El nuevo síntoma introducido en la configuración de *anaflow* consta del conjunto de patrones mostrados en la tabla 3.5. Sin embargo, existe un problema, y es que el proceso *anaflow* desarrollado hasta el momento está fuertemente orientado a literales de texto y no soportaba patrones binarios, como son los que nos interesan en este caso. Por tanto, fue necesario desarrollar un nuevo tipo de patrón (HSTR) que permitiera el uso de este tipo de patrones, tarea que se detalla en 6.2.

Afortunadamente, la secuencia de audio MP3 está alineada a octeto, con lo que se simplifica la detección de patrones enormemente. Sin embargo, los patrones de interés no están formados por un número entero de bytes, ya que constan de 21 bits.

Aunque se pensó en la posibilidad de desarrollar un mecanismo de detección de patrones binarios bit a bit (configurándolos con el uso de máscaras, por ejemplo), se desestimó la posibilidad[28], ya que implicaría modificaciones complejas en *anaflow*. Por tanto, hemos de utilizar patrones que incluyan también los bits de CRC, relleno y privado, cubriendo todas las posibilidades en los mismos.

El resultado es que, aunque realmente estamos interesados solo en 4 patrones de 21 bits, al tener que considerar 3 bytes (24 bits) tenemos que configurar un total de 32 en rule.cfg para conseguir todas las combinaciones, que son las que se muestran en la tabla 3.8.

Obsérvese que además de los patrones para la detección MP3, se ha incluido un nuevo síntoma, DUMMY, que consta de cuatro patrones de 3 bytes (igual longitud que los patrones MP3). Su utilidad se verá en el apartado 3.5.4.

La configuración de *vflow* resulta más sencilla. Tan solo hay que añadir el nuevo síntoma MP3 a la lista de síntomas de tipo lúdico de verdict.cfg (figura 2.10).

SYM;LUD;IRC,SEX,DROGAS,JUEGOS,SPORT,MUSICA,**MP3**

3.5.4 Evaluación de resultados

Una vez configurada la plataforma se pasó a una evaluación de la efectividad de los nuevos heurísticos introducidos para la detección MP3, cuyo resultado se describe en el presente apartado.

Los resultados analizados se corresponden a trazas de utilidad de patrones extraídas a lo largo del primer mes de funcionamiento de la plataforma con

[28] De todas formas, se plantea como posible trabajo futuro, en 7.2.6.

#formato 128 Kbps - 44.1 kHz
MP3;HSTR;FFFA90;
MP3;HSTR;FFFA91;
MP3;HSTR;FFFA92;
MP3;HSTR;FFFA93;
MP3;HSTR;FFFB90;
MP3;HSTR;FFFB91;
MP3;HSTR;FFFB92;
MP3;HSTR;FFFB93;
#formato 160 Kbps - 44.1 kHz
MP3;HSTR;FFFAA0;
MP3;HSTR;FFFAA1;
MP3;HSTR;FFFAA2;
MP3;HSTR;FFFAA3;
MP3;HSTR;FFFBA0;
MP3;HSTR;FFFBA1;
MP3;HSTR;FFFBA2;
MP3;HSTR;FFFBA3;
#formato 192 Kbps - 44.1 kHz
MP3;HSTR;FFFAB0;
MP3;HSTR;FFFAB1;
MP3;HSTR;FFFAB2;
MP3;HSTR;FFFAB3;
MP3;HSTR;FFFBB0;
MP3;HSTR;FFFBB1;
MP3;HSTR;FFFBB2;
MP3;HSTR;FFFBB3;
#formato 24 Kbps - 11.025 kHz
MP3;HSTR;FFE230;
MP3;HSTR;FFE231;
MP3;HSTR;FFE232;
MP3;HSTR;FFE233;
MP3;HSTR;FFE330;
MP3;HSTR;FFE331;
MP3;HSTR;FFE332;
MP3;HSTR;FFE333;
#patrones aleatorios
DUMMY;HSTR;ED558D;
DUMMY;HSTR;0857C9;
DUMMY;HSTR;3DB543;
DUMMY;HSTR;996101;

Tabla 3.8: Configuración para la detección de patrones MP3.

la nueva configuración. Dichas trazas fueron obtenidas gracias a las nuevas funciones de medida de uso (cuyos detalles de diseño e incorporación se detallan en el apartado 6.3).

En concreto, se han analizado un conjunto de 5 trazas, elegidas de entre las más representativas del periodo[29]. En la tabla 3.9 se muestran los resultados de estas trazas (y del agregado).

A parte de los resultados correspondientes propiamente a patrones MP3, en esta tabla también se ha incluido lo siguiente:

- El síntoma DUMMY consta de cuatro patrones aleatorios de 3 bytes y lo que se pretende con él es ver una medida del "ruido" presente en la clasificación por patrones. Lo usaremos para comprobar como es de bueno un patrón MP3[30]. Si un patrón MP3 tiene un nivel de ocurrencias menor o similar al de un patrón DUMMY, resulta igual de útil que un patrón aleatorio para detectar flujos, y por tanto, debería ser eliminado, puesto que no sirve de mucho.

- También se incluye el número de ocurrencias del síntoma HTTP. Normalmente, este es el síntoma que mayor número de ocurrencias tiene de todas las que existen en rule.cfg, y nos servirá para comparar el éxito de los patrones de detección MP3.

El análisis de estos resultados lleva a las siguientes conclusiones:

- En primer lugar, observamos el sorprendentemente alto número de éxitos que tienen los patrones MP3, con unas cifras no solo comparables sino en ocasiones mayores que las del síntoma HTTP (que hasta ahora siempre habían sido las más altas).

- Por otra parte, se observa que no todos los 32 patrones resultan igual de exitosos. Algunos de ellos se encuentran por debajo o muy próximos al umbral de utilidad, que podemos fijar, dentro de una traza, como el máximo número de ocurrencias en el conjunto de patrones DUMMY (entre 191 y 348 en los casos de la tabla 3.9). En concreto, se pueden establecer las siguiente comparaciones.

[29] Es decir, aquellas correspondientes a invocaciones de *anaflow* que duraron más tiempo. No hemos de olvidar que, debido a los procesos de vigilancia de la plataforma, *anaflow* puede ser reiniciado con cierta aleatoriedad

[30] Aplicamos esta comparación para comprobar la utilidad de los MP3, pero en realidad podría utilizarse para comprobar la utilidad de cualquier tipo de patrón.

Patrón	Traza 1	Traza 2	Traza 3	Traza 4	Traza 5	Total
FFFA90	1319	2399	1120	1350	2007	8195
FFFA91	866	1047	1103	1096	1309	5421
FFFA92	12646	28427	9652	10369	25496	86590
FFFA93	1809	1866	8975	605	5216	18471
FFFB90	145307	204173	116053	116041	213252	794826
FFFB91	338	820	1165	821	2788	5932
FFFB92	262097	254603	311860	229286	479252	1537098
FFFB93	373	7797	13823	8588	25136	55717
FFFAA0	495	724	1551	1156	1115	5041
FFFAA1	360	456	315	434	1258	2823
FFFAA2	301	465	782	847	655	3050
FFFAA3	310	476	347	466	887	2486
FFFBA0	42074	28642	35648	28718	54870	189952
FFFBA1	6480	1067	1168	360	912	9987
FFFBA2	24110	17036	19542	14750	26057	101495
FFFBA3	4428	935	1032	351	876	7622
FFFAB0	521	1278	744	1065	3488	7096
FFFAB1	622	868	1018	657	1248	4413
FFFAB2	1190	11370	1414	8909	44448	67331
FFFAB3	313	408	5225	907	421	7274
FFFBB0	96823	39368	29426	27697	238054	431368
FFFBB1	896	1364	2140	1745	1291	7436
FFFBB2	81878	100794	76078	74033	317141	649924
FFFBB3	9496	13973	22488	21102	17397	84456
FFE230	237	455	268	326	544	1830
FFE231	179	361	204	298	432	1474
FFF232	249	443	213	271	378	1554
FFF233	196	320	195	248	244	1203
FFF330	308	304	1880	946	1644	5082
FFF331	159	283	137	187	498	1264
FFF332	146	250	158	153	492	1199
FFF333	171	239	181	265	430	1286
Total MP3	**696697**	**723011**	**665905**	**554047**	**1479236**	**4118896**
ED558D	168	203	134	231	230	966
0857C9	129	185	99	131	312	856
3DB543	156	174	80	129	378	917
996101	250	292	191	279	348	1360
Total DUMMY	**703**	**854**	**504**	**770**	**1268**	**4099**
Umbral	**250**	**292**	**191**	**279**	**348**	**3451525**
HTTP	**1362342**	**263768**	**423726**	**191952**	**1209737**	**3451525**

Tabla 3.9: Resultados obtenidos con los patrones MP3.

- En lo referente al audio de calidad (patrones FFFA y FFFB), presentan un mayor número de éxitos los que tienen a 1 el bit de CRC (FFFB) frente a los que no lo tienen (FFFA). Es decir, la tendencia que parece existir en los MP3 que circulan por la red es no utilizar protección CRC (tendencia que ya hemos señalado en varias ocasiones a lo largo de este capítulo).
- Otra tendencia que parece darse es la de tener desactivado el bit de privado, observando comparativamente el éxito de las reglas *****{0,2} frente a las de tipo *****{1,3}.
- De entre los tres tipos de audio de calidad (128, 160, 192 Kbps) el formato que resulta más popular es el de 128 Kbps, que cosecha un número de éxitos significativamente mayor al de los otros dos.
- Las aplicaciones que transmiten audio de baja calidad (mayoritariamente las de *streaming*) no parecen estar siendo muy utilizadas en la actualidad, a la vista del poco éxito de los patrones dispuestos para detectar este tipo de tráfico.

En vista de estos resultados, podemos considerar modificaciones en la configuración de *anaflow* que llevan a una mejora en su eficiencia. De los 32 patrones configurados inicialmente, eliminamos aquellos que no resultan demasiado útiles (pocas ocurrencias) y nos quedamos con los más significativos. En concreto, podemos utilizar una nueva configuración con los 6 patrones que se muestran en la tabla 3.10.

Patrón	Traza 1	Traza 2	Traza 3	Traza 4	Traza 5	Total
FFFB90	145307	204173	116053	116041	213252	794826
FFFB92	262097	254603	311860	229286	479252	1537098
FFFBA0	42074	28642	35648	28718	54870	189952
FFFBA2	24110	17036	19542	14750	26057	101495
FFFBB0	96823	39368	29426	27697	238054	431368
FFFBB2	81878	100794	76078	74033	317141	649924
Total	**652289**	**644616**	**588607**	**490525**	**1328626**	**3704663**

Tabla 3.10: Nueva configuración de patrones MP3.

Con esta reducción de patrones, el número total de ocurrencias disminuye de 4118896 a 3704663 (aproximadamente en un 10%), que es aceptable si consideramos que la disminución de patrones de 32 a 6 (reducción del 81.25%), repercutirá en un incremento significativo de la eficiencia de *anaflow*, al tener que inspeccionar menos patrones dentro de los contenidos.

Por otra parte, observar que dentro de estos patrones no se encuentra ninguno de los especializados en *streaming*. A la vista de los resultados anteriores, dichos patrones han tenido muy poco éxito y, por tanto, no queda justificada su inclusión. No obstante, es previsible que esta tendencia cambie si se produce el auge definitivo de este tipo de aplicaciones, por lo que es un aspecto que convendría vigilar en el futuro.

Al margen de los resultados obtenidos con las trazas de utilidad, también se realizó un análisis de la agregación horaria de ficheros AFL que realiza *accum* (apartado 2.3.1.3), donde se observó que los flujos en los que aparece el síntoma MP3 presentan una gran cantidad de ellos (muchas veces del orden del centenar), frente a otro tipo de síntomas (HTTP, JUEGOS o DEPORTE), en los que el número de síntomas no suele superar la decena. Esto se debe a que los patrones están basados en las cabeceras de las tramas MP3 y a que estas se producen con cierta regularidad durante el flujo (aspecto que ya se vio en el estudio previo), lo que no ocurre en general con otro tipo de patrones de otros síntomas. Desde el punto de vista del sistema de clasificación de *vflow*, esto es muy positivo, ya que difícilmente se equivocará a la hora de clasificar basándose en mayoría de síntomas, con lo que los flujos LUD que se correspondan a MP3 van a ser clasificados con gran exactitud.

Por último, como resultado de alto nivel, en los informes de resultados presentados a RedIris desde que se incorporaron las nuevas técnicas de detección de MP3, se observa un gran incremento en el porcentaje medio de tráfico lúdico, lo cual es un dato significativamente positivo y que respalda el éxito de las medidas adoptadas.

3.6 Conclusiones

El fuerte incremento percibido en el tráfico lúdico no hace sino demostrar la evolución en los hábitos de los usuarios en el uso de las redes de comunicación, en un principio pensadas para fines académicos y corporativos y en las cuales ahora mismo existe una gran demanda de contenidos lúdicos. Es necesario contar con métodos para monitorizar estos comportamientos, y debido a ello, el estudio y desarrollo de técnicas especializadas en la detección de tráfico lúdico (como el presente trabajo) cobra una gran importancia.

De entre las dos filosofías de detección en las que pueden basarse estas técnicas (la propia estructura de los contenidos o las aplicaciones y protocolos que los transmiten) llegamos a la conclusión de que son más idóneas las primeras. No solo porque el diseño de la propia aplicación dificulte la detección (por ejemplo, al usar puertos aleatorios) sino porque están muy sujetas a modas volubles. No compensa el esfuerzo de investigación del protocolo de una nueva aplicación P2P si es altamente probable que pasados unos pocos meses los usuarios dejen de utilizarla y las técnicas desarrolladas basadas en ella se vuelvan inútiles.

La propia historia del P2P nos aporta ejemplos. En 1999, Napster era el líder indiscutible, debido a su ventaja por ser el primer sistema P2P en aparecer. Hoy en día, existe todo un mosaico de aplicaciones, con continuas bajas y nuevas apariciones (AudioGalaxy, iMesh, WinMX, . . .).

Los formatos de fichero han resultado ser bastante más estables y por tanto, compensa la investigación destinada a ellos. MP3 se ha consolidado *de facto* como estándar de audio comprimido por Internet. Sin embargo, tampoco hay seguridad de que vaya a permanecer para siempre, y habría que estar atento a nuevos formatos (y al "redescubrimiento" de antiguos) que constituyen alternativas emergentes: RealAudio, OGG, WMA, AAC, ...

Otra de las conclusiones que ha revelado este trabajo es que hoy en día, a pesar de que ya existen aplicaciones para el usuario final, el *streaming* no están muy difundido. Sin embargo, es de esperar que con la mejora de los accesos (ADSL, cable modem, GPRS, . . .), esta tendencia cambiará en el futuro, hasta el punto que puede que no dentro de mucho tiempo se invierta la situación con respecto a las aplicaciones de intercambio de ficheros. ¿Quién necesita almacenar un fichero en su ordenador para escucharlo cuando puede tenerlo donde quiera y en el momento que quiera en tiempo real?

Este aumento en la capacidad del acceso también llevará previsiblemente a un aumento en el consumo de contenidos multimedia. Si bien durante algún tiempo los usuarios se han conformado con el audio, las aplicaciones P2P más recientes incluyen la posibilidad de intercambio de vídeos. En particular, a pesar de su polémico origen, el formato [divX] está alcanzando gran popularidad y es muy posible que se convierta en el "fenómeno MP3" del futuro.

Capítulo 4

Funciones de seguridad

4.1 Introducción

Una de las principales características de Internet es que se trata de una red de comunicación completamente abierta, a la que en principio cualquier entidad puede estar conectada. Este factor, que por una parte es una ventaja pues ha permitido el crecimiento espectacular de la Red a lo largo de su historia, también conlleva una desventaja en lo que se refiere a la seguridad.

Internet es una red insegura, donde cualquier usuario, desde el cliente residencial de un ISP hasta la empresa dotada de red privada corporativa propia, está expuesto a ataques de seguridad procedentes del exterior por el simple hecho de estar conectado. El riesgo de sufrir uno de estos incidentes ha de ser tenido muy en cuenta, ya que el impacto que puede existir si efectivamente algún ataque se produce con éxito es considerable, pudiendo llegar a cuantiosas pérdidas económicas en el peor de los casos.

Además, la seguridad en Internet es un campo en continua evolución. Las técnicas de ataque avanzan constantemente, así como los mecanismos de seguridad destinados a evitar dichos ataques, con lo que en definitiva nos encontramos en una frenética carrera de evolución que enfrenta por un lado a los atacantes de sistemas[1] y por otro a los expertos en seguridad.

En este escenario, una plataforma de captura y análisis de tráfico como MIRA puede resultar de gran utilidad. Por una parte, permite al consultor estar al tanto de la actividad de ataques de seguridad presente en la red monitorizada. En función de esta información el consultor podrá tomar las acciones administrativas que considere oportunas. Por otra parte, registrar la actividad

[1]Utilizamos el término "atacante" para referirnos al sujeto que origina un ataque de seguridad en la red. Comúnmente se utilizan también los términos anglosajones *cracker* y, más discutido, *hacker*.

de los atacantes de seguridad puede resultar de gran utilidad para investigar que técnicas de ataque se utilizan y desarrollar mecanismos de protección frente a las mismas.

Por todo lo anterior, el desarrollo de funciones de seguridad y su incorporación efectiva en la plataforma MIRA, que son el objeto del presente capítulo, quedan completamente justificados. El primer paso será la definición de nuestros objetivos, es decir, que tipo de funcionalidad queremos incorporar en la plataforma MIRA en lo referente a seguridad (apartado 4.2). A continuación, realizaremos una descripción de la herramienta Snort, en algunos aspectos muy similar a la plataforma MIRA, ya que nos será de gran utilidad en la incorporación de funciones de seguridad (apartado 4.3). Seguidamente, definiremos cual será la arquitectura elegida (apartado 4.4) y como se ha implementado (apartado 4.5). Por último, finalizaremos este capítulo con las conclusiones derivadas del trabajo realizado (apartado 4.6).

4.2 Definición de objetivos

4.2.1 Seguridad lógica

Por gestión de seguridad de red se entiende el conjunto de acciones de planificación, organización, supervisión y control necesarias sobre equipos de comunicación (redes, servicios, sistemas y aplicaciones) que garanticen la protección de los mismos frente a incidentes dañinos (entendido como tal todo aquel que provoca pérdidas, ya sea de equipos, datos, disponibilidad de servicio, daños en la imagen corporativa, ...).

Esta definición es muy amplia y la protección ha de cubrir muchos aspectos. Podemos distinguir dos grandes áreas de seguridad.

- Seguridad *física*, cuyos objetivos son la protección de los equipos en sí frente a amenazas del entorno (alarmas de incendio, unidades de alimentación ininterrumpida, protección frente a sobretensiones, ...) y garantizar que solo las personas adecuadas acceden físicamente a los mismos (vigilancia presencial, llaves de acceso, alarmas volumétricas, ...)

- Seguridad *lógica*, cuyos objetivos son la protección del acceso lógico a los equipos[2] y de la información en tránsito que circula entre distintas redes por un entorno potencialmente inseguro.

[2] O más bien, a los servicios, aplicaciones y datos que estas manejan, ya que preferimos reservar el término "equipo" para el sistema físico.

Nos centraremos en seguridad lógica, ya que la seguridad física es un área completamente ajena al trabajo de desarrollo de la plataforma que estamos realizando en este documento.

Desde el punto de vista del administrador de una red, la realización de una gestión de seguridad es un proceso que consta de las siguientes etapas.

1. Identificar y valorar que es lo que se quiere proteger. Es decir, determinar cuales son los recursos susceptibles de ataque. Típicamente, estos pueden incluir:

   - Equipos físicos
   - Utilización de equipos (CPU, ancho de banda, ...)
   - Información confidencial
   - Software
   - Información en tránsito
   - Imagen corporativa

2. Identificar las amenazas. Se consideran tres aspectos: origen, probabilidad e impacto.

   En lo que respecta al origen, si dejamos aparte las amenaza de entorno (cubiertas por la seguridad física) el mayor origen de amenazas son las propias personas, por una parte debido a errores de utilización de los recursos y por otra (más grave) debido a ataques malintencionados (tanto internos como externos). Aunque la intencionaidad es muy diferente, ambos tipos de amenaza pueden considerarse bajo el mismo enfoque, al ser sus efectos bastante similares[3]. Asumiremos en lo sucesivo ataques malintencionados.

   Sobre las razones que llevan a los atacantes a actuar como lo hacen, muchas veces se encuentran detrás de ellas motivos económicos (robo de datos bancarios, espionaje industrial, ...), pero otras, en un porcentaje significativo, es tan solo una cuestión de "desafío personal". Ciertamente, no resulta fácil explicar las motivaciones que hay detrás de muchos ataques y no es el objetivo de este trabajo profundizar en ello. A nuestros efectos, simplemente asumimos que tales ataques se producen, sin entrar en las causas últimas de los mismos.

[3] Al fin y al cabo, tiene el mismo efecto la caída de un servidor web debido a que un usuario provocó un desbordamiento en el mismo consultando una página web que debido a un atacante malintencionado que conoce y explota dicha vulnerabilidad, por poner un ejemplo.

La caracterización de una amenaza se realiza a través de la *probabilidad* (medida de la potencialidad de la amenaza) y el *impacto* (pérdidas que se producen si la amenaza se materializa, ya sean pérdidas económicas, tiempo, imagen, ...) de la misma. El producto de ambos factores es lo que se denomina *exposición al riesgo* o, simplemente *riesgo*[4].

$$\text{Riesgo} = \text{Probabilidad} \times \text{Impacto}$$

Hay que destacar que la determinación precisa de probabilidad e impacto (y por tanto, también el riesgo) es muy complicada, imposible la mayoría de las veces. Por ello, se confía en valoraciones subjetivas que, al menos, permiten una ordenación relativa de las distintas amenazas ('muy vulnerable', 'poco vulnerable', ...).

3. Definición de una política de seguridad. La *política de seguridad* es un documento que describe el conjunto de acciones que la organización va a llevar a cabo para disminuir los riesgos de seguridad. Se definen dos tipos de medidas, cada una de ellas actuando sobre uno de los factores del riesgo:

   - Medidas curativas. Destinadas a disminuir el impacto.
   - Medidas preventivas. Destinadas a disminuir la probabilidad.

   La definición de la política de seguridad se realiza en función al análisis de amenazas anterior. Hemos de elegir que amenazas vamos a cubrir, ya que la seguridad implica unos costes (equipos y personal especializado, restricciones y *overhead* en la prestación de servicios, mantenimiento, ...) y los recursos económicos que podemos dedicar son limitados. Nos interesará cubrir aquellas amenazas que supongan un mayor riesgo, mientras que aquellas cuyo coste de impacto sea menor que el coste de seguridad para cubrirlas no han de ser tenidas en cuenta.

4. Ejecución y mantenimiento de la política de seguridad. No solo es importante la implementación de la política de seguridad, sino la monitorización de la misma, mediante informes, registros y alarmas. Como ya señalamos en la introducción, la seguridad lógica es un área que evoluciona muy rápidamente y conviene estar lo más al día posible. Además,

[4]Realmente, esto tiene aplicaciones más amplias que la seguridad de red, pudiendo utilizarse en general en todos aquellos problemas de planificación donde existan factores de incertidumbre con un efecto en dicha planificación, como por ejemplo la gestión de proyectos de ingeniería o la estrategia militar.

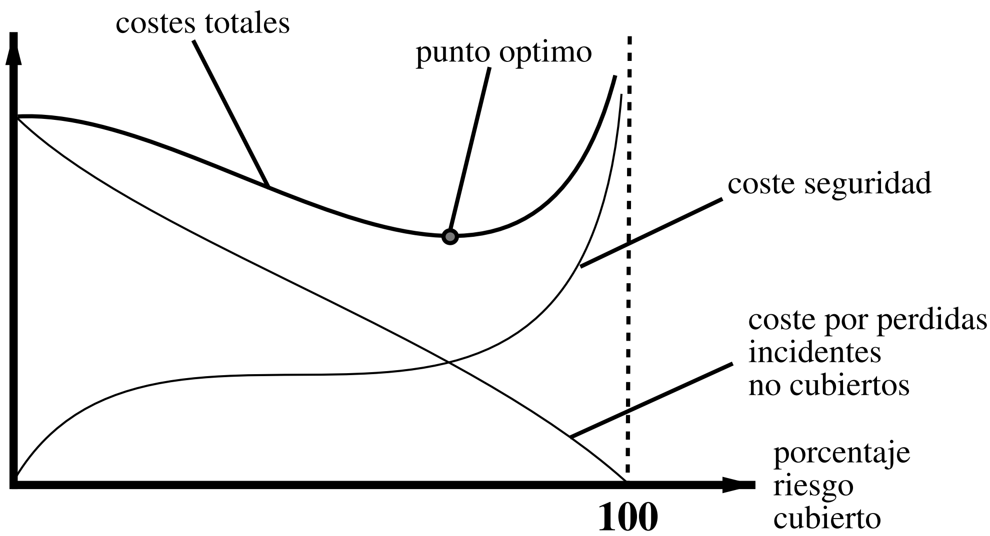


Figura 4.1: Costes de la seguridad.

los registros de actividad sospechosa tienen una gran importancia, pues pueden ser usados para tomar acciones de denuncia.

Como ya hemos mencionado en la introducción al capítulo, la seguridad lógica es un campo de actividad muy amplio. Puede consultarse como referencias de tipo general [Stallings 95], [Ahuja 96] y [Forcht 94] para profundizar en el tema. Por otra parte, [Cheswick+ 94] toma un enfoque práctico de la seguridad de red, mientras que [Davies+ 94] se centra en aspectos criptográficos.

4.2.2 Planteamiento de la seguridad

Una vez expuesto el marco de gestión de la seguridad en un entorno de red, vamos a analizar que posibilidades de aplicación presenta la plataforma MIRA. Dado que se trata de una plataforma de captura y análisis de tráfico, es obvio que su utilidad va a encontrarse en la monitorización de la seguridad.

Sin embargo, no hemos de olvidar que la monitorización no es sino la última de una serie de etapas que definen el proceso de gestión de seguridad explicado en el apartado 4.2.1. Conviene reflexionar de forma separada para ambos escenarios de aplicación de la plataforma (descritos en el apartado 2.4). En escenarios de red local el proceso de gestión de seguridad se adapta perfectamente a lo explicado. Los activos a proteger están claros: los recursos de la red local que estamos monitorizando.

En el escenario de red troncal ya no queda tan claro que es lo que estamos intentado proteger. En primer lugar podría pensarse que lo que protegemos son los recursos de la red troncal, como por ejemplo los routers, conmutadores y demás elementos, incluida la propia plataforma de monitorización. Sin embargo, en el plano de red troncal, estos elementos se limitan a redirigir tráfico, por lo que los potenciales ataques se producirán en el plano de su gestión. El acceso a los equipos para su gestión no se realiza normalmente mediante la propia red troncal, sino mediante redes locales asociadas a los centros de gestión del operador. Es por tanto el escenario de red local es que habría que considerar para la protección de los recursos propios, no el de red troncal.

Otra posibilidad es que el objetivo sea la protección de los sistemas administrativos conectados a la red troncal. Sin embargo, este enfoque plantea un problema relacionado con la responsabilidad del consultor de la plataforma. El operador de red troncal garantiza un servicio de interconexión a los sistemas administrativos pero, ¿debe ocuparse también de la protección de los mismos o es esta una responsabilidad de los propios sistemas administrativos? Si bien ambas podrían ser posturas válidas en principio, la realidad es que se tiende más hacia lo segundo.

No obstante, en ciertos casos puede tener sentido reforzar las acciones de seguridad de los sistemas administrativos con funciones de seguridad en la red troncal, especialmente si esta red es de tipo público, como RedIris.

Para entender las diferencias entre ambos escenarios, podemos plantear el siguiente símil. Podemos pensar en la red troncal como las calles de una ciudad, que interconectan urbanizaciones de residentes, equivalentes a los sistemas administrativos. Al igual que los sistemas finales tienen su propia gestión de la seguridad, las urbanizaciones cuentan con vigilancia privada. Pero también existe una policía pública, encargada de la vigilancia en las calles, equivalente a las funciones de seguridad desplegadas en la red troncal. Ambos cuerpos tienen funciones específicas, pero pueden (y deberían) colaborar para combatir la delincuencia.

De todas formas, toda la discusión anterior queda en el plano administrativo más que en el plano técnico, ya que las funciones de seguridad de la plataforma MIRA como herramienta de monitorización se adaptan a cualquier caso. Esta flexibilidad es debida a que, ya sea en un escenario o en otro, las amenazas de seguridad son las mismas. Siguiendo con el símil, MIRA es un "arma" que podrá ser empleada tanto por vigilantes privados como por policías para perseguir a unos ladrones que son del mismo tipo en ambos casos.

4.2.3 Objetivos técnicos

Los ataques de seguridad se producen de forma remota[5]. Por tanto, un atacante necesitará intercambiar tráfico de paquetes con el sistema atacado a través de la red monitorizada. Este es un tráfico que podemos capturar y analizar con la plataforma y de esa manera detectar el ataque.

Por tanto, los objetivos de seguridad que nos planteamos son los siguientes:

- Incorporar funciones de análisis que permitan detectar ataques de seguridad dentro del tráfico capturado, en base a perfiles de actividad sospechosa detectados sobre dicho tráfico. Denominamos *firma* de un ataque al conjunto de características de tráfico (patrón dentro del contenido de paquetes, puertos que utiliza, ...) que identifican ese ataque.

- Estudiar e implementar distintas formas de registrar dichos ataques de seguridad y notificarlos al consultor. Principalmente, estamos interesados en mecanismos de monitorización pasiva, en la línea de la monitorización de tráfico habitual realizada por la plataforma MIRA, pero también estaremos interesados en incorporar ciertas funciones activas (que serán objeto de estudio en el apartado 4.4.4), útiles en algunos escenarios de aplicación.

- Realización de los dos objetivos anteriores con el mínimo impacto sobre las funciones preexistentes de clasificación de tráfico convencional.

Existe una diferencia entre ambos escenarios de aplicación, de tipo técnico. Como ya se razonó en 2.4, en el escenario de red local era posible realizar la captura de todo el tráfico cursado por la red, mientras que en el de red troncal no podíamos utilizar esa suposición. Por tanto, en este segundo caso, es posible que aquellos ataques que impliquen interacciones complejas y mucho intercambio de tráfico no sean detectados. Hemos de tener en cuenta y asumir esta limitación[6].

[5] Esto no es necesariamente cierto para todos los casos. Por ejemplo, un atacante que esté dentro de las instalaciones del operador de red y acceda a la consola de uno de sus equipos podría realizar un ataque no remoto. Sin embargo, la prevención de este tipo de acciones entra más dentro de la seguridad física, la cual, como ya se dijo en 4.2.1, no consideraremos en este trabajo.

[6] Forzando nuestro símil del apartado anterior, la ciudad es demasiado grande y la policía no puede estar en todas partes.

4.3 Snort

En el presente apartado vamos a analizar la herramienta de seguridad Snort. En concreto, la versión que hemos analizado y utilizado en este trabajo es la 1.8.4. Este software es de libre distribución y puede descargarse desde Internet.

Si se desea ampliar la información aquí presentada puede consultarse la descripción que hace de Snort su propio autor en [Roesch 99].

4.3.1 Introducción a Snort

Snort es un NIDS o *Network Intrusion Detection System* (sistema de detección de intrusiones de red). Los autores hablan de dos modos más de funcionamiento, *sniffer* y *packet logger* (registrador de paquetes), pero en realidad son dos casos particulares de funcionalidad reducida del anterior.

El trabajo que realiza la herramienta consiste en escuchar un flujo de paquetes (típicamente, los que captura por una interfaz de red o un filtrado de los mismos), analizarlos y (en función del resultado de este análisis) tomar una acción: ignorar el paquete, registrarlo o generar una alerta (al generar una alerta, también se produce el registro del paquete).

Como vemos, las funciones de captura y análisis de tráfico son muy similares a MIRA, si bien Snort se especializa en la detección de tráfico sospechoso de ataques de seguridad.

Realmente, Snort es tan solo un eficiente motor de captura y análisis de tráfico, que no sabe por si mismo como tiene que analizar los paquetes. Es el usuario quien se lo indica, mediante un fichero de configuración. El fichero de configuración está compuesto de un conjunto de reglas (con una sintaxis y semántica concretas) preparadas para identificar las firmas de los ataques. Cada vez que se captura un paquete, este se compara con las reglas, de forma que si dispara alguna de ellas se toma la acción correspondiente que se haya configurado. Nos detendremos más detenidamente en la configuración de Snort en el apartado 4.3.3.

Snort está fuertemente orientado a redes de área local. De hecho, se basa en la librería `libpcap` para la captura de los paquetes[7]. Podemos distinguir dos modos de funcionamiento, según la forma en la que se recibe el flujo de paquetes a analizar.

- *Online.* En este caso el programa se ejecuta en segundo plano indefinidamente (hasta que el usuario le manda una señal para que se detenga), escuchando una (o varias) interfaces de red del host (con posibilidad de

[7]Esta librería es la misma que utilizan conocidas herramientas de análisis como [tcpdump] o [Ethereal].

escuchar en toda la red, si se desea, configurando la tarjeta en modo promiscuo) y procesando los paquetes según le van llegando.

- *Offline*. El flujo de paquetes ha sido capturado previamente en un fichero de captura en el formato de libpcap. Snort lo utiliza como entrada, analiza el flujo en él grabado y genera sus resultados.

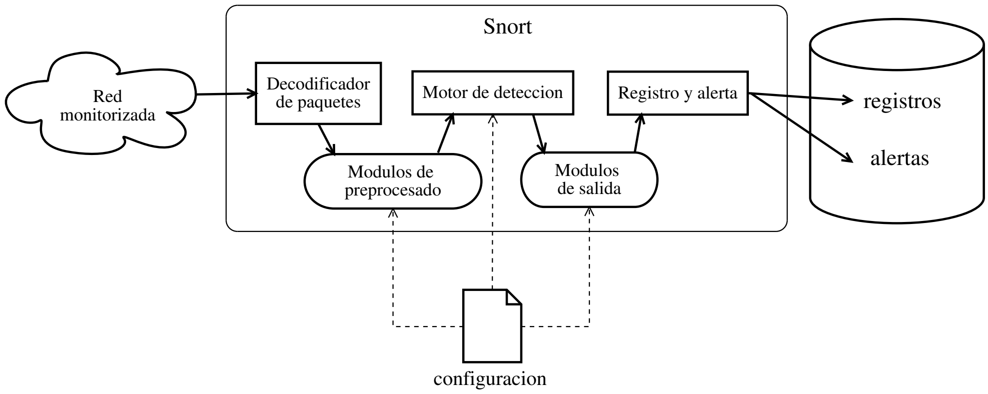


Figura 4.2: Arquitectura interna de Snort.

La figura 4.2 muestra la arquitectura básica de Snort[8]. Los bloques principales son el decodificador de paquetes (cuya tarea es decodificar el paquete según los distintos protocolos de la pila TCP/IP), el motor de análisis (encargado de realizar la comprobación de reglas) y la función de registro y alerta (que se encarga de producir la salida del programa).

Si bien lo explicado en el párrafo anterior se corresponde con el funcionamiento básico de Snort, existen funcionalidades adicionales, incorporadas en las últimas versiones.

- Comportamiento reactivo. Si bien Snort está diseñado como una herramienta de monitorización pasiva, es posible también configurarlo para que reaccione ante determinadas situaciones, convirtiéndose de esa forma en un elemento activo dentro de la infraestructura de seguridad. Por ejemplo, puede cortar una conexión TCP en la que se haya detectado actividad sospechosa (enviando segmentos FIN a ambos extremos).

- Módulos de preprocesado. Estos módulos realizan un procesado sobre el flujo de paquetes capturados, antes de entregarlos al analizador. De esta

[8]Funcionando en modo *online*. En modo *offline* es igual, salvo que en vez de extraer la captura de tráfico de una red, se hará desde un fichero.

manera, se pueden añadir capacidades de análisis basadas en el estado de un flujo de paquetes, permitiendo de esta forma la detección de ataques complejos (exploración de puertos o detección de tráfico en puertos anómalos, por ejemplo). Analizaremos los módulos de preprocesado en el apartado 4.3.4.

- Módulos de salida. Con estos módulos pueden añadirse capacidades avanzadas en la salida que proporciona el programa que van más allá de la notificación de alertas y registro de paquetes en ficheros de texto plano. Por ejemplo, se puede mandar la alerta a una base de datos. Analizaremos los módulos de preprocesado en el apartado 4.3.5.

Estos módulos (tanto los de preprocesado como los de salida) no forman parte del motor de Snort en sí, que los carga a modo de *plug-ins*. De hecho, están escritos por diversos autores, siendo este sistema una manera de hacer Snort más modular y facilitar así el desarrollo de nuevas funcionalidades. Hay que tener también en cuenta que la utilización de módulos añade una carga adicional de proceso que puede mermar la eficiencia de Snort.

De Snort podemos destacar las siguientes características como ventajas:

- Flexibilidad. Es adaptable a gran número de plataformas y situaciones. De hecho, el presente trabajo es una prueba. Gran parte de la flexibilidad de Snort se debe a que el usuario puede configurar la mayor parte de sus funciones.

- Ligero. El consumo de recursos de Snort (capacidad de proceso y memoria, básicamente) es pequeño, cuando se lo compara con el consumo de otros sistemas de detección de intrusiones, que en ocasiones requieren el uso de plataformas dedicadas. La eficiencia de Snort depende de muchos factores (potencia del hardware donde se ejecuta, configuración de la herramienta, ...), pero como figura de mérito, se ha llegado a utilizar hasta en redes Ethernet 100 Mbps con un nivel de saturación de 80 Mbps.

- Bajo coste. Al ser una herramienta de libre distribución, bajo licencia [GNU 89], su coste de adquisición es nulo. Esto contrasta con los elevados precios de los sistemas de detección de intrusiones comerciales, cuyo precio ronda los miles de euros en el mejor de los casos y puede llegar a los cientos de miles en los casos más extremos.

En los siguientes apartados, se van a detallar aspectos concretos de la herramienta y su configuración. Información adicional de los mismos puede encontrarse en [Roesch 02].

4.3.2 Alerta y registro de paquetes

Se puede configurar el modo de alerta deseado, entre seis posibilidades.

- Alerta rápida. La alerta consiste en marca de tiempo, mensaje de alerta y origen y destino (IP y puerto).

```
02/04-14:25:22.525676  [**] [1:1113:1] WEB-MISC http directory traversal
[**] [Classification: Attempted Information Leak] [Priority: 2]
{TCP+} 192.168.251.162:1765 -> 192.168.1.106:80
```

- Alerta completa. Modo por defecto. La alerta consiste en la misma información que en el modo de alerta rápida añadiendo información de las cabeceras del paquete.

```
[**] [1:1113:1] WEB-MISC http directory traversal [**]
[Classification: Attempted Information Leak] [Priority: 2]
02/04-14:25:22.525676 213.116.251.162:1765 -> 172.16.1.106:80
TCP TTL:111 TOS:0x0 ID:11031 IpLen:20 DgmLen:496 DF
***AP*** Seq: 0x8E406992  Ack: 0x2CAE9E9B  Win: 0x2238  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS297]
```

- Socket UNIX. Envía la alerta a un *socket* UNIX que otro programa puede estar escuchando.

- Syslog. La alerta se registra en el syslog (mecanismo de sistemas operativos UNIX para registrar incidencias del sistema) para luego ser procesada por otros programas.

- SMB. Mediante este mecanismo puede enviarse alertas WinPopup a clientes Windows.

- Ninguna. Alertas desactivadas.

Conviene también describir a este punto la forma básica[9] en la que Snort genera las alertas y los registros de paquetes. Para ello, Snort utiliza una serie de ficheros, que se localizan en un directorio que se especifica como parte de la configuración (por defecto, se usa el directorio /var/log/snort), como ilustra la figura 4.3.

- Fichero de alertas (alert). Se trata de un fichero en ASCII plano donde se registran las alertas.

[9] Hablamos de salida básica, porque esto puede complicarse debido al uso de los módulos de salida que se describen en el apartado 4.3.5.

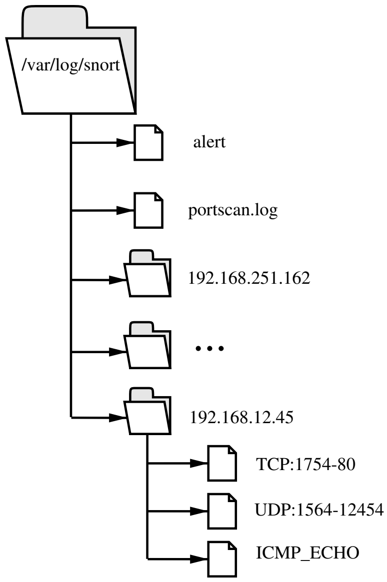


Figura 4.3: Directorio de registro y alertas de Snort (por defecto).

- Ficheros de registro. Los registros se encuentran estructurados en un árbol de subdirectorios basado en la dirección IP del paquete registrado (IP destino relativo a una subred origen, si se especifica; en caso contrario, IP destino u origen de forma aleatoria), dentro del cual el nombre del fichero indica el protocolo y los puertos. El formato de estos registros es un fichero ASCII plano (con una representación hexadecimal del contenido del paquete).

  Es significativo también el caso de registro en forma de fichero binario. Si se usa esta opción (equivalente al módulo de salida log_tcpdump descrita en 4.3.5) no se usa un árbol de directorios: todos los paquetes registrados se almacenan en un único fichero binario en el formato de libpcap, al que se le añade en el nombre una marca de tiempo del instante (minuto y segundo) en el que se creó dicho fichero.

- El uso de preprocesadores (apartado 4.3.4) puede producir la generación de ficheros adicionales. Por ejemplo, el módulo portscan puede generar un fichero (portscan.log) con registro de las exploraciones de puertos producidas.

4.3.3 Configuración de reglas

La configuración se especifica en un fichero (cuyo nombre típico es snort.conf), que Snort lee al arrancar. Existe también la posibilidad de distribuir la configuración en varios ficheros, que se enlazan desde el principal mediante la directiva include. Esto permite organizar las reglas de forma modular (por ejemplo, el fichero dns.rules con reglas para detectar ataques al servicio DNS, el fichero netbios.rules con reglas para ataques a Netbios, ...) y conseguir con ello una gran flexibilidad. Así, por ejemplo, si en nuestra red no tenemos servidores de DNS podemos desactivar ese conjunto de reglas y aplicar en todo caso una configuración adaptada a nuestra situación.

El fichero de configuración contiene tres tipos de elementos: variables, directivas y reglas, de los cuales nos centraremos en estas últimas.

- Variables. Pueden definirse variables, cuyo nombre puede utilizarse dentro de las reglas en sustitución del valor al que representan.

```
var MY_NET [192.168.1.0/24,10.1.1.0/24]
```

- Directivas. Configuran aspectos de Snort que no son propiamente reglas. Por ejemplo, nombre del directorio de registro, máscara de creación de ficheros, ...

- Reglas. Cada regla consta de dos partes: una cabecera y una parte de opciones. En la cabecera se especifican acción, protocolo y direcciones origen y destino. En la parte de opciones (que no es obligatoria) se especifica información adicional sobre la regla y condiciones que debe cumplir un paquete, teniendo que cumplirse todas ellas para que la regla se dispare[10].

A continuación vamos a detallar los campos de los que consta la parte de cabeceras y de opciones de una regla.

```
alert tcp any any -> 192.168.1.0/24 111 (content:"|00 01 86 a5|"; \
   msg: "mountd access";)
```

[10]Puede verse como un AND lógico, así como el conjunto de reglas que forman la configuración se pueden ver como un OR lógico.

4.3.3.1 Cabecera

Acción. Posibles acciones[11]:

- alert. Genera una alerta (según el modo elegido) y registra el paquete.
- log. Registra el paquete, pero no produce alerta.
- pass. Ignora el paquete.

A parte de estas posibilidades por defecto, el usuario también puede definir sus propios subtipos de regla, mediante la directiva `ruletype` a partir de un tipo básico (alert o log) asociado a uno o varios módulos de salida. Un ejemplo de esto puede encontrase en 4.3.5.

Las reglas se aplican siguiendo un orden: primero las reglas de alerta, luego las de paso y, por último, las de registro, si bien es posible cambiarlo en la configuración. Si el usuario está utilizando reglas que son subtipos de otras, estos se aplican en el orden en el que se definieron (dentro de cada uno de los tipos)[12].

Protocolo. La versión actual de Snort soporta ip, icmp, tcp y udp. No obstante, mediante la opción ip_proto, puede especificarse cualquier protocolo que pueda encapsularse sobre IP.

Direcciones IP. Se pueden especificar direcciones IP aisladas y subredes. También se pueden expresar combinaciones y negaciones de las mismas. Se puede utilizar el literal 'any' para indicar que cualquier dirección activa la regla.

Además de indicar las direcciones ha de definirse el sentido (es decir, cual de las direcciones debe interpretarse como origen del paquete y cual es el destino) con el operador `->`. Si la interpretación del origen y destino es irrelevante, se usa el operador de bidireccionalidad `<>`.

Puerto. Cuando el protocolo es TCP o UDP se indican los puertos (origen y destino). Pueden indicarse puertos concretos, rangos, negaciones o utilizar el literal 'any' para indicar que el puerto es irrelevante.

[11] A parte de las tres señalas, existen dos acciones más activate y dynamic, pero en la versión actual su funcionalidad es reemplazada por la de la opción tag.

[12] Por ejemplo, si en el fichero de configuración se definieron los subtipos alert1 y alert2 (de tipo alert) y log1 y log2 (de tipo log) en el orden: alert1, log1, alert2, log2, el orden de aplicación de las reglas serán: alert1, alert2, log1, log2.

4.3.3.2 Opciones

Ninguna de las opciones es obligatoria. Se especifican entre paréntesis, tras la cabecera de la regla, separando las distintas opciones con punto y coma (;). Conviene tener presente las características de los protocolos de Internet (IP, TCP, UDP, ICMP, RPC, ...) para comprender estas opciones. Para ello, puede consultarse [Stevens 94] como referencia de la pila de protocolos TCP/IP.

msg. Imprime un mensaje en alertas y registros de paquetes. Su utilidad es puramente informativa.

logto. Registra el paquete en un fichero distinto al especificado como salida estándar.

ttl. Comprobación del valor del tiempo de vida en la cabecera IP.

tos. Comprobación del valor del tipo de servicio en la cabecera IP.

id. Comprobación del valor ID en la cabecera IP. Algunas herramientas de ataque utilizan valores específicos en este campo.

ipoption. Comprobación de la presencia de opciones en la cabecera IP: registro de ruta, marca de tiempo, encaminamiento fijado en el origen, ...

fragbits. Comprobación de los bits de fragmentación en la cabecera IP: reservado (RB), más fragmentos (MF) y no fragmentar (DF).

dsize. Comprobación del tamaño del contenido (carga útil) del paquete. Se puede especificar un valor exacto o una relación (mayor o menor que). Es de gran utilidad para la comprobación de desbordamientos (*overflows*) de buffer[13], comprobando que el tamaño de los paquetes no excede el máximo que utiliza determinado servicio. La ventaja es que esta forma de detección es mucho más rápida que una inspección de contenidos.

flags. Comprobación de los *flags* en la cabecera TCP: FIN, SYN, RST, PSH, ...

seq. Comprobación del número de secuencia en la cabecera TCP.

ack. Comprobación del asentimiento en la cabecera TCP.

[13] Técnica de ataque que consiste en enviar a un servidor un paquete mayor de lo habitual, de forma que el exceso desborda el buffer de lectura, instalando en el servidor un código malicioso.

itype. Comprobación de tipo de paquete en cabecera ICMP. En ocasiones ataques de negación de servicio por inundación[14] utilizan paquetes ICMP de tipos inválidos, que pueden ser detectados fácilmente con esta opción.

icode. Comprobación del código de paquete en la cabecera ICMP. Similar a itype.

icmp_id. Comprobación del campo ICMP ECHO ID. Algunos troyanos de *backorifice*[15] utilizan túneles ICMP para evitar los cortafuegos. Estos túneles pueden ser detectados por ciertos valores característicos en este campo.

icmp_seq. Comprobación del número de secuencia de un paquete ICMP ECHO. La utilidad es la misma que la de la opción anterior.

content. Una de las opciones más importantes. Realiza la búsqueda de un determinado patrón dentro del contenido (carga útil) del paquete, resultando en éxito si lo encuentra. Esta búsqueda se realiza empleando el algoritmo de Boyer-Moore (descrito en el apéndice D), uno de los más eficientes que existen para la realización de esta tarea.

Por defecto, se distingue mayúsculas y minúsculas al realizar la búsqueda. También pueden utilizarse patrones que son en parte o completamente binarios (expresados en hexadecimal, con el carácter de escape '|'). Varias opciones content pueden usarse en la misma regla.

content-list. Búsqueda de un conjunto de patrones en el contenido (carga útil) del paquete, que se especifican dentro de un fichero.

offset. Modificador a la opción de content. Indica en que punto de la carga útil ha de comenzarse la búsqueda del patrón.

depth. Modificador a la opción content. Indica que extensión de la carga útil (a partir del offset) ha de inspeccionarse en la búsqueda del patrón.

```
alert tcp any any -> 192.168.1.0/24 80 (content: "cgi-bin/phf"; \
   offset: 3; depth: 22; msg: "CGI-PHF access";)
```

[14] Técnica de ataque que consiste en el envío masivo de paquetes sin sentido, para colapsar la capacidad de respuesta de un servidor, de forma que el servicio que este presta quede interrumpido.

[15] Un *backorifice* es un servidor oculto, que se instala de forma no desea en un host, de forma que el atacante puede acceder a él para obtener información del sistema y controlarlo remotamente.

nocase. Modificador a la opción content. Indica que no se haga distinción entre mayúsculas y minúsculas en el patrón.

regex. Modificador de content. Permite utilizar expresiones regulares al especificar el patrón. Los autores avisan que todavía es una funcionalidad en desarrollo y que no debería usarse en entornos de producción.

session. Permite imprimir los datos de usuario de sesiones TCP. De esta forma, puede registrarse lo que un usuario de Telnet, FTP o web está enviando o recibiendo. La opción tiene dos valores posibles: printable (registra solo aquello que el usuario puede ver o teclear) o all (registra todo los datos de sesión, sustituyendo los caracteres no imprimibles por sus equivalentes en hexadecimal).

Hay que tener en cuenta que el registro de una sesión puede disminuir considerablemente la eficiencia de Snort, puesto que supone una gran carga de proceso.

rpc. Decodifica peticiones RPC y comprueba la aplicación, procedimiento y versión. Se considera éxito cuando los tres campos encajan. Se pueden usar comodines en los campos de procedimiento y versión.

resp. Indica una respuesta que debe ser tomada por Snort cuando un paquete dispara la regla. Puede tomarse una o varias respuestas. Las posibilidades son distintas alternativas de cierre de conexión TCP o envío de paquetes ICMP.

react. Similar a resp, pero la respuesta es a más alto nivel. Está pensado principalmente para utilizar con el servicio web, de forma que un cliente accediendo a contenidos inapropiados (por ejemplo, pornografía) reciba un aviso en su navegador, con la posibilidad incluso de bloquear la conexión.

Mediante las opciones resp y react se incorporan funciones de respuesta activa en Snort, como ya se comentó en 4.3.

reference. Incluye una referencia a un sistema externo de identificación de ataques, que puede ser utilizada por el administrador para obtener más información sobre como actúa el ataque que la regla pretende detectar.

sid. Identificador de regla.

rev. Número de revisión de la regla.

Las opciones sid y rev no afectan al modo en el que la regla se evalúa, pero establecen un mecanismo de mantenimiento mediante el cual puede ser gestionado y actualizado el conjunto de reglas.

classtype. Permite clasificar las reglas por grupos, según su nivel de peligrosidad. Las categorías por defecto (que se encuentran configuradas en el fichero classification.config) se establecen en función de tres niveles de prioridad: alta (1), media (2) y baja (3), siendo más peligrosos los ataques que detectan las reglas de mayor prioridad. Las tablas 4.1, 4.2 y 4.3 muestran las distintas categorías, según su índice de prioridad.

Categoría	**Descripción**
attempted-admin	Intento de obtener privilegios de administrador
attempted-user	Intento de obtener privilegios de usuario
shellcode-detect	Detectado código ejecutable
successful-admin	Privilegios de administrador obtenidos con éxito
successful-user	Privilegios de usuario obtenidos con éxito
trojan-activity	Troyano de red detectado
unsuccessful-user	Fallo en obtener privilegios de usuario
web-application-attack	Ataque a aplicación web

Tabla 4.1: Categorías de prioridad alta.

priority. Permite establecer la prioridad de una regla directamente. En el caso de que esa regla pertenezca a una categoría con una prioridad distinta (classtype), toma preferencia la opción priority.

uricontent. Realiza una búsqueda de contenidos, pero solo en el fragmento de paquete correspondiente al URI, ignorando el resto del contenido. De esta forma, se evitan falsos positivos[16] en la detección de ataques a servidores web.

tag. Permite registrar paquetes adicionales al que cumple la regla, asociados al mismo, aunque estos paquetes subsiguientes no la disparen. Se pueden registrar paquetes asociados a la misma sesión TCP o al host que disparó la regla (origen o destino). También se especifica cuantos paquetes o durante cuanto tiempo hay que registrar el tráfico adicional.

[16] Un falso positivo es un evento registrado como alerta de seguridad cuando realmente no lo es. Es decir, se trata de un error de la medida.

Categoría	**Descripción**
attempted-dos	Intento de negación de servicio
attempted-recon	Intento de pérdida de información
bad-unknown	Tráfico potencialmente dañino
denial-of-service	Detección de ataque de negación de servicio
misc-attack	Ataque misceláneo
non-standard-protocol	Detección de un protocolo o evento no estándar
rpc-portmap-decode	Decodificación de una petición RPC
successful-dos	Denegación de servicio
successful-recon-largescale	Pérdida de información a gran escala
successful-recon-limited	Pérdida de información
suspicious-filename-detect	Nombre de fichero sospechoso detectado
suspicious-login	Intento de login usando nombre de usuario sospechoso
system-call-detect	Llamada a sistema detectada
unusual-client-port-connection	Cliente usando puerto no usual
web-application-activity	Acceso a una aplicación web potencialmente peligrosa

Tabla 4.2: Categorías de prioridad media.

Categoría	**Descripción**
icmp-event	Evento ICMP genérico
misc-activity	Actividad miscelánea
network-scan	Detección de exploración de red
not-suspicious	Tráfico no sospechoso
protocol-command-decode	Comando de protocolo genérico decodificado
string-detect	Cadena sospechosa detectada
unknown	Tráfico desconocido

Tabla 4.3: Categorías de prioridad baja.

Esta opción resulta útil para la investigación de ataques, ya que no solo se registra el paquete que lo inició, sino también aquellos que el atacante utiliza posteriormente.

ip_proto. Comprueba el campo de protocolo encapsulado en el datagrama IP (siguiendo la numeración que se especifica en [RFC 1700]).

sameip. Comprueba si la dirección origen y destino del paquete es la misma.

stateless. Utilizado por el módulo de preprocesado Stream4, para indicar reglas que no deben de integrarse dentro de la detección basada en estados. Ver descripción de este preprocesador en 4.3.4 para más detalles.

Conviene destacar a este punto que el orden en el que se evalúan las reglas no coincide necesariamente con el orden en el que especifican en los ficheros de configuración. Esto es debido a la forma en la que Snort construye sus árboles de decisión.

La estructura utilizada es una lista encadenada bidimensional con dos tipos de nodos: cabeceras de cadena (*chain headers*) y las opciones de cadena (*chain option*). Reglas con una misma cabecera son agrupadas bajo una misma cabecera de cadena, con las distintas posibles opciones de cadena colgando de ella. La comprobación de un paquete se realiza recorriendo las cabeceras de cadena y, dentro de cada una de ellas, las opciones de cadena.

Por ejemplo, consideremos la siguiente configuración:

```
alert tcp any any -> $HOME 80 (content: "uno"; msg: "UNO";)
alert tcp any any -> $HOME 1:1024 (flags: S; msg: "DOS";)
alert tcp any any -> $HOME 80 (flags: S; content: "tres"; msg: "TRES";)
alert tcp any any -> $HOME 80 (content: "cuatro"; msg: "CUATRO";)
```

Esto daría lugar a la estructura de reglas que se muestra en la figura 4.4. Observar que el orden de evaluación de las reglas es UNO, TRES, CUATRO, DOS, a diferencia de lo que podría pensarse intuitivamente por la posición de las reglas en la configuración.

Otra importante consideración a tener en cuenta es que la opción content es la comprobación computacionalmente más costosa de realizar. Esto se ha tenido en cuenta en el diseño de Snort, de forma que en un *chain options* si existen opciones adicionales a content estas se comprueban primero. Si alguna de ellas ya es falsa, no se evalúa la opción content. Podemos explotar este hecho incluyendo reglas con opciones lo más ricas posibles (por ejemplo, usar siempre que se pueda la opción flags, como en TRES).

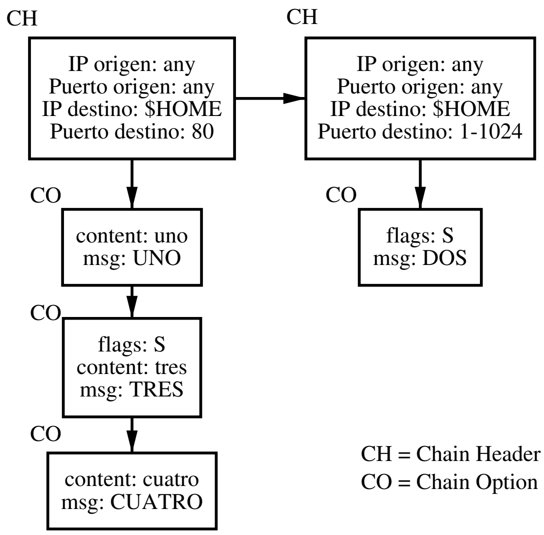


Figura 4.4: Ejemplo del árbol de decisión usado en Snort.

4.3.3.3 Ficheros de reglas que acompañan a la distribución

Dentro de la distribución de Snort se incluye un conjunto de ficheros de reglas especializadas en ciertos ataques de seguridad. Esto resulta muy conveniente, ya que nos evita el arduo trabajo de investigación que sería analizar los cientos de ataques de seguridad existentes y extraer reglas para detectarlos.

El conjunto de reglas de Snort no es estático, sino que se actualiza constantemente mediante las contribuciones de sus desarrolladores. El papel de la comunidad de investigación es muy importante en este aspecto, pues como ya se ha señalado anteriormente constantemente se encuentran agujeros de seguridad para los cuales han de desarrollarse reglas de detección. En ocasiones, el tiempo que pasa desde que se publica un incidente de seguridad[17] hasta que los desarrolladores de Snort proporcionan una regla para detectarlo es de tan solo una pocas horas.

El conjunto de reglas puede actualizarse mediante CVS. A modo de ejemplo, describimos los principales ficheros que acompaña a la distribución 1.8.4 (y que contiene unas 1200 reglas aproximadamente), aunque, como se ha señalado, este conjunto va cambiando con cada nueva versión.

exploit.rules contiene reglas que aprovechan ataques contra vulnerabilidades conocidas del software de los servidores, típicamente ataques de desbordamiento.

[17] Para ello existen foros especializados, como los [CERT] o las listas de correo BugTraq.

shellcode.rules contiene reglas que identifican código maligno y que se comprueban en todo el tráfico (con un significativo impacto en la eficiencia de Snort).

backdoor.rules contiene reglas para detectar la presencia de servidores ocultos de *backorifice*.

dos.rules contiene reglas que identifican intentos de negación de servicio (*Denial of Service* o *DoS*).

ddos.rules contiene reglas que identifican intentos de negación de servicio distribuida (*Distributed DoS*)[18].

web-attacks.rules contiene reglas para la detección de ataques contra servidores web.

web-cgi.rules contiene reglas para detectar ataques contra vulnerabilidades del CGI de los servidores web.

web-iis.rules contiene reglas para detectar ataques a las vulnerabilidades del Internet Information Server de Microsoft.

web-coldfusion.rules contiene reglas para detectar ataques a ColdFusion.

web-frontpage.rules contiene reglas para detectar ataques contra vulnerabilidades de Frontpage de Microsoft.

web-misc.rules contiene reglas que identifican otros ataques a servidores web.

netbios.rules contiene reglas que detectan ataques contra las vulnerabilidades del protocolo Netbios.

rservices.rules contiene reglas para detectar ataques contra servicios de login remoto (como rlogin y rsh).

ftp.rules contiene reglas para detectar ataques al servicio FTP.

tftp.rules contiene reglas para detectar ataques al servicio TFTP.

finger.rules contiene reglas para detectar ataques al servicio de Finger.

smtp.rules contiene reglas para detectar ataques contra el servicio de correo SMTP.

[18] La diferencia con respecto a un ataque de DoS no distribuido es que en este caso colaboran un conjunto de atacantes cooperativamente para colapsar el servidor del servicio.

dns.rules contiene reglas para detectar ataques contra el servicio de DNS.

rpc.rules contiene reglas para detectar ataques contra el servicio de RPC.

telnet.rules contiene reglas para detectar ataques contra el servicio de Telnet.

x11.rules contiene reglas para detectar ataques al protocolo de terminal gráfico remoto X11.

sql.rules contiene reglas para detectar ataques al servicio SQL.

icmp.rules (junto con **icmp-info.rules**) contiene reglas para detectar tráfico sospechoso de paquetes ICMP, que pueden ser indicios de ataques.

attack-responses.rules contiene reglas que identifican respuestas de algunos servicios ante ciertos ataques de seguridad.

scan.rules contiene reglas para detectar exploraciones de redes, típicamente exploraciones de puertos, mapeo IP y aplicaciones de *scanning*.

virus.rules contiene reglas especializadas en detectar ciertos tipos de virus y troyanos.

bad-traffic.rules contiene reglas para identificar tráfico erróneo, en el sentido de que son paquetes que nunca deberían presentarse en una red y que, en tal caso, son indicio de actividad sospechosa (por ejemplo, paquetes con el TTL a 0 o con igual dirección IP origen que destino).

misc.rules contiene reglas que no encajan bien en ninguna de las otras categorías.

4.3.4 Módulos de preprocesado

La configuración de los módulos de preprocesado se realiza en el mismo fichero de configuración de reglas (puede verse como un tipo especial de directivas). En el apéndice C.1 pueden encontrarse ejemplos. A continuación destacamos los más importantes.

- Decodificador HTTP (http_decode). Procesa las URI de las cabeceras HTTP convirtiendo los caracteres codificados en x-www-form-urlencoded (es decir, los codificados como %XX) a su equivalente en ASCII plano. Esto permite detectar ataques que intentan engañar a Snort utilizando este tipo de caracteres en las peticiones HTTP sin tener que introducir nuevas reglas de detección de patrones. Similarmente, existe un preprocesador decodificador de RPC (rpc_decode) y otro de Telnet (telnet_decode) que se encargan de normalizar el tráfico que utilizan estos protocolos.

- Detector de backorifice (bo). Realiza un procesado del tráfico para la detección de servidores de *backorifice*. Este tráfico muchas veces va encriptado y este preprocesador incluye mecanismos de desencriptación para conseguir detectarlo.

- Detector de exploración de puertos[19] (portscan). Genera una alerta cada vez que se detecta una exploración de puertos, definida esta como el envío de paquetes TCP SYN a más de P puertos en menos de T segundos (o el envío de paquetes UDP a más de P puertos en menos de T segundos) por parte de una misma fuente a la misma o distintas máquinas destino, siendo P y T parámetros configurables por parte del usuario. Adicionalmente, si se especifica, puede registrar destino de la exploración de puertos, puertos explorados y tipo de exploración en un fichero (típicamente portscan.log).

- Defragmentador IP (frag2). Realiza reensamblado de datagramas IP antes de entregarlos al sistema de detección. De esta forma, se evita que los atacantes puedan engañar a Snort fragmentando los paquetes a nivel IP con un tamaño de datagrama muy pequeño[20].

- Reensamblado TCP (Stream4). Conceptualmente, es similar al reensamblado IP. Sin embargo, en este además se mantiene un estado de los flujos TCP[21]. Gracias a ello, es posible configurar Snort para que solo alerte de trafico TCP en flujos que hayan sido establecidos siguiendo el mecanismo *three way handshake* del protocolo (descrito en [RFC 793]) o en los que se haya observado actividad cooperativa bidireccional.

  Con este preprocesador se evitan ataques de confusión contra Snort, como los realizados por herramientas como [Snot], [Stick] y [Fragroute]. Estos ataques consisten en el envío de paquetes que son inocuos pero que poseen la firma de ataques conocidos, con la intención de desconcertar el

[19] Una exploración de puertos (*port scanning*) es una técnica de descubrimiento de servicios que consiste en enviar paquetes a todos los puertos de un host para determinar cuales están abiertos. Suele preceder al ataque en sí, que utilizará vulnerabilidades en el servicio de alguno de los puertos abiertos.

[20] Piénsese, por ejemplo, en que pasaría si el ataque se realizara utilizando datagramas IP de 1 byte de datos y no se usara este preprocesador.

[21] Téngase en cuenta, que a diferencia de IP o UDP, TCP es un protocolo con estado, en el que las etapas de establecimiento de la conexión, transmisión de datos y liberación de la conexión están bien definidas.

sistema de detección y el registro de alertas con falsos positivos. Puesto que las herramientas que realizan este tipo de ataques no se preocupan en efectuar el *three way handshake*, Snort ignorará estos paquetes.

Existe la posibilidad de indicar en una regla que se ignore completamente la inspección de estado, mediante la opción stateless. Puede resultar conveniente utilizarla para evitar el elevado consumo de recursos de este preprocesador, ya que mantener el estado de múltiples conexiones TCP, puede suponer un significativo impacto en la eficiencia de Snort. Adicionalmente, se permite al usuario configurar algunos aspectos del funcionamiento interno del módulo, como por ejemplo puertos sobre los que se debe realizar el reensamblado, tiempo de vida de una sesión inactiva, límite de memoria que debe usar el módulo, ...

- Detección de tráfico anómalo (Spade). Este módulo permite la detección de actividad anómala. Por tanto, puede resultar muy útil para el descubrimiento de nuevos ataques mediante la detección del tráfico en puertos que se supone no deberían de cursarlo. Para ello, utiliza un modelo basado en teoría de información (con cálculo de entropias) y umbrales de alerta adaptativos.

  Puede encontrase información más detallada de este preprocesador en [Spade].

Algunos de estos preprocesadores también pueden producir alertas por sí mismos. Estas pueden diferenciarse claramente de las alertas estándar de Snort, puesto que van precedidas del literal ‘spp’ (acrónimo de *Snort Pre-Processor.*)

4.3.5 Módulos de salida

Los módulos de salida permiten capacidades avanzadas en la presentación de la salida de Snort (registro y alertas). La utilización de estos módulos se realiza mediante la definición de subtipos de reglas con ruletype con uno o varios módulos de salida asociados. Si se especifican varios, los módulos se aplican en cascada, uno detrás de otro.

```
ruletype example_alert
{
   type alert output
   alert_syslog: LOG_AUTH LOG_ALERT
   output database: log, mysql, user=snort dbname=snort host=localhost
}
```

Vamos a mencionar a continuación los principales módulos de salida.

- Módulos de alerta alert_syslog, alert_fast, alert_full, alert_unixscock, alert_smb). Equivalentes a los modos de alerta especificados en 4.3.2.
- Registro binario (log_tcpdump). Mediante este módulo, el registro de paquetes se realiza en un fichero, en el formato de libpcap (descrito en el apéndice A.2). Este fichero puede ser analizado posteriormente por gran número de herramientas (como [tcpdump], [Ethereal] o incluso el propio Snort funcionando en modo *offline*).
- XML (xml). Con este módulo se genera una salida en lenguaje SNML (*SNort Markup Language*), que es un lenguaje XML con su propia DTD.
- Base de datos (database). Permite enviar la salida de Snort a una gran variedad de bases de datos SQL. El uso de una base de datos para registrar eventos de seguridad resulta muy interesante ya que por una parte permite centralizar la recogida de alertas por parte de múltiples sensores Snort y por otra permite desacoplar el proceso de notificación de alertas (Snort envía su salida a la base de datos) del de consulta de las mismas (peticiones realizadas por aplicaciones independientes a la base de datos).

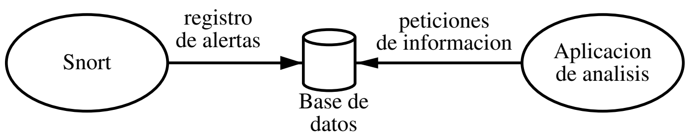


Figura 4.5: Uso de Snort con base de datos.

- SNMP (trap_snmp). Mediante este módulo, Snort puede informar de alertas a una estación de gestión de red (*Network Management Station* o NMS) mediante notificaciones con el protocolo de gestión SNMP. Puede resultar interesante el uso de esta capacidad para aquellas alertas que, dada su peligrosidad, resulten críticas y tengan que ser notificadas inmediatamente a los administradores de la red monitorizada.

4.3.6 Herramientas relacionadas con Snort

En torno a Snort se han desarrollado un conjunto de herramientas software que trabajan cooperativamente con él para complementar y mejorar capacidades o adaptarlas a situaciones específicas. Entre ellas, podemos destacar herramientas de generación de informes (que procesan la salida de Snort y la presentan de una forma elaborada), interfaces para bases de datos y módulos de integración con servidores web (como por ejemplo módulos PHP).

La descripción de todas y cada una de estas herramientas nos llevaría una extensión considerable dentro de este trabajo. Recomendamos al lector una consulta a [Snort] para obtener más información sobre las mismas. No obstante, en la implementación de las funciones de seguridad en la plataforma MIRA haremos uso de dos de ellas, [SnortSnarf] y [Guardian], que describiremos a continuación.

4.3.6.1 SnortSnarf

La herramienta SnortSnarf es un *script* programado en Perl que resulta muy útil para la generación de páginas web basadas en los registros de Snort. A continuación se va a explicar el funcionamiento básico de esta herramienta, que nos proporcionará informes HTML planos, para luego ir comentando las funciones más avanzas que permite. Paralelamente, se explicarán los pasos de configuración de esta herramienta.

Asumimos la existencia de un servidor web (por ejemplo Apache), a cuyo directorio de ficheros tenemos acceso, por ejemplo en el punto /var/www/html/snort. Resulta conveniente (aunque no obligatorio) que dicho directorio esté protegido, de forma que solo las personas autorizadas puedan acceder a la información de seguridad a través de la web.

Suponemos también que Snort genera sus alertas y registros de paquetes en el directorio /var/log/snort.

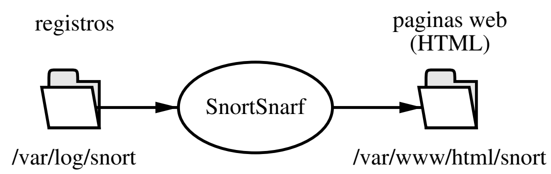


Figura 4.6: Funcionamiento de SnortSnarf.

Generación de informes básicos La forma más sencilla de usar SnortSnarf es la siguiente (tras haber instalado convenientemente los módulos que acompañan a la distribución del programa, tal y como se indica en la documentación correspondiente).

```
snortsnarf.pl -d /var/www/html/snort/ \
   /var/log/snort/alert /var/log/snort/portscan.log
```

Con ello, generamos un informe HTML cuya página de inicio es /var/www/html/snort/index.html, y con la información basada en los ficheros de alerta alert (alertas estándar de Snort) y portscan.log (si usamos el preprocesador de exploración de puertos) especificados. Observar que en este caso se utilizarán como entradas los ficheros alert y portscan.log, pero en realidad podemos indicar más ficheros (por ejemplo, para el caso en el que hayamos divido las alertas en varios ficheros en función de su peligrosidad o para integrar alertas producidas por distintas máquinas.).

La estructura del informe consta de tres secciones.

- Sección de firmas. En la página principal se muestra un resumen de todas las alertas registradas. Mediante un enlace podemos acceder a información más detallada y a referencias externas que proporcionan información sobre el ataque (opción reference de Snort). La información detallada de cada alerta consiste en una lista de orígenes y destinos que han originado/recibido la alerta.

- Ranking de orígenes y destinos de alerta. Muestra una clasificación de los hosts IP que más ataques han realizado o que más ataques han recibido (esto resulta muy interesante, tanto para descubrir posibles nidos de atacantes como para descubrir las víctimas más atacadas). El tamaño de esta lista es 20 por defecto para ambos casos (aunque puede configurarse a otro valor con -top).

- En las tres secciones existen enlaces a orígenes IP y a destinos IP, que muestran la misma información pero organizada en base a direcciones en vez de a alertas. En cada página, se muestra que alertas se han recibido y el detalle de las mismas (la cabecera del paquete, básicamente).

Trabajar con direcciones IP resulta mucho más incomodo que trabajar con nombres de dominio. Es posible invocar a SnortSnarf para que consulte al DNS al generar las páginas y de esa forma, siempre que pueda, proporcionará el nombre de del host además de su dirección IP (parámetro -dns). Sin embargo, hay que tener cuidado con esto: es posible que haya que consultar cientos de direcciones, lo cual (a parte de implicar un elevado retardo al crear las páginas) puede colapsar los servidores de DNS (al producirse un gran número de peticiones en muy corto espacio de tiempo). Como figura de mérito, la generación de un informe típico puede tardar unos 10 segundos si no se realizan consultas al DNS, mientras que si se utilizan, este tiempo asciende a más de 20 minutos.

La alternativa a esto es realizar la consulta al DNS no en el momento de generar las páginas sino durante la visualización, mediante enlaces integrados por SnortSnarf en las páginas con servidores de DNS de uso público. Existen facilidades adicionales: también hay enlaces de whois (ARIN, RIPE, ...), si queremos información administrativa sobre determinada dirección IP (lo cual puede resultar muy útil, por ejemplo, para avisar a los administradores de una red infectada) y consulta a base de datos de puertos (para obtener más información acerca de un determinado puerto TCP o UDP).

Generación de informes avanzados Si bien los informes anteriormente descritos muestran la información en un modo bastante más cómodo que la consulta plana de ficheros de registro, es posible enriquecerlos con más información.

En concreto, podemos realizar las siguientes configuraciones:

- Incluir referencias a las reglas de Snort

  ```
  snortsnarf.pl -d /var/www/html/snort/ -rulesfile snort.conf \
     /var/log/snort/alert /var/log/snort/portscan.log
  ```

  Siendo snort.conf el fichero de configuración de reglas de Snort. De esta forma, en la sección de alertas, se incluye la descripción la regla (en sintaxis Snort) que ha producido esas alertas. Puede resultar útil para depuración de reglas.

- Incluir enlaces a los ficheros de registro de paquetes. En las páginas donde se muestran cabeceras de paquete, es posible acceder al paquete completo mediante un enlace que nos lleva al fichero de registro de Snort. Al invocar SnortSnarf hemos de indicarle donde encontrar estos registros, mediante una URL.

  ```
  snortsnarf.pl -d /var/www/html/snort/ \
     -ldir "http://localhost.localdomain/snort/log" \
     /var/log/snort/alert /var/log/snort/portscan.log
  ```

  Con este ejemplo, indicamos que la URL donde se encontrarán los registros es http://localhost.localdomain/snort/log. Para que funcionen correctamente estos enlaces, hemos de situar el directorio de los logs dentro del servidor web. Si los logs se encuentran en otro sitio, basta con generar un enlace simbólico[22], por ejemplo:

[22] Hay que tener precaución con esto porque el servidor web puede estar configurado para no permitir enlaces simbólicos dentro de su árbol de directorios. Por ejemplo, en servidores Apache, ha de usarse la opción FollowSymLinks.

```
ln -s /var/log/snort /var/www/html/snort/log
```

Hemos de tener precaución con los permisos de los ficheros para que todo funcione, ya que Snort genera los ficheros y directorios de forma que solo puede acceder a ellos el propietario de los mismos (700 para directorios y 600 para ficheros) y lo que queremos es que sean accesibles para el proceso del servidor web (755 para directorios, 644 para ficheros). La utilidad fix_perms.pl que acompaña la distribución de SnortSnarf resulta útil para ello. Otra posibilidad es indicarle a Snort una máscara de creación de ficheros distinta de la que utiliza por defecto (parámetro -m).

- Filtrar alertas en función de la prioridad. Por defecto, SnortSnarf incluye en sus informes todas las alertas que encuentra en los ficheros de registro de alertas. Esto, en determinados casos, puede generar tal cantidad de información que abrume al consultor de los informes. Sin embargo, es posible configurar SnortSnarf para que incluya en el informe solo las alertas que sobrepasen un determinado nivel de prioridad (indicado con las opciones priority o classtype de Snort).

```
snortsnarf.pl -d /var/www/html/snort/ -minprio=2 \
   /var/log/snort/alert /var/log/snort/portscan.log
```

  Con este ejemplo, estamos indicando que la prioridad mínima que queremos mostrar es 2 (-minprio=2), con lo que no se mostrarán alertas de prioridad baja (3). Observar que aquellas alertas que no tienen ningún valor de prioridad asignado son mostradas siempre.

- Filtrar alertas en función de la red. Podemos indicar a SnortSnarf que incluya en el informe aquellas alertas que provengan (con -sipin) o vayan hacia (con -dipin) una subred determinada. De esta forma, podemos centrarnos en el análisis de seguridad de unas determinadas redes, eliminando las alertas generadas por otras subredes (y que, en la práctica, puede resultar en "ruido" más que en información para el consultor).

Bases de datos de anotaciones Para potenciar el uso de informes y dar un mayor valor añadido al HTML plano, es posible integrarlos con una base de datos de anotaciones en XML. Para ello, es necesario, en primer lugar, que el servidor web soporte CGIs, ya que la creación y visualización de anotaciones en esta base de datos se producirá a través de *scripts* ejecutados por el propio servidor. Por otra parte, necesitamos tener convenientemente instalados los módulos Perl necesarios para realizar análisis de XML (y que serán utilizados por los *scripts*).

Observar que no ha de confundirse esta funcionalidad con la posibilidad de comunicación directa con bases de datos del propio Snort mediante el uso del módulo de salida adecuado (figura 4.5). En este último caso, Snort produce su salida hacia una base de datos mientras que la solución planteada con SnortSnarf se basa en HTML, aunque, como añadido, proporcione un limitado uso de base de datos de anotaciones de seguridad para facilitar la labor al usuario de la web.

Hemos de crear una base de datos vacía, con los correspondiente permisos de lectura y escritura para que el servidor web (a través de los CGIs) pueda acceder a ella. Para ello, puede utilizarse la utilidad setup_anns_dir.pl que acompaña a la distribución de SnortSnarf. Asumimos que la base de datos de anotaciones se encuentra en /var/log/XML. También hemos de ubicar los *scripts* en el directorio correspondiente de los CGIs.

```
snortsnarf.pl -d /var/www/html/snort/ -db /var/log/XML/annotations.xml \
   /var/log/snort/alert /var/log/snort/portscan.log
```

Si el directorio donde se encuentran los CGIs es otro distinto a http://localhost.localdomain/cgi-bin, hemos de especificar la URL con el parámetro -cgidir para que SnortSnarf genere enlaces correctos a los scripts.

Podemos ver y añadir anotaciones de tres tipos:

1. Asociadas a una dirección IP.
2. Asociadas a una subred (con 24 o 16 bits de máscara).
3. Asociadas a un tipo de alerta.

Cada anotación consta de cuatro campos:

1. Nombre del creador.
2. Fecha (automático).
3. Asunto.
4. Texto de la anotación en sí.

A parte de poder realizar anotaciones a través de enlaces situados en las propias páginas HTML, es posible realizar una búsqueda por dirección IP, subred o tipo de alerta a través de un formulario situado en la página principal, sin la necesidad de tener que navegar hasta la página del host IP o tipo de alerta concreto.

Sin embargo, este sistema tiene algunas limitaciones:

- No podemos utilizar caracteres especiales al escribir las anotaciones (caracteres acentuados, letra ñ, ...). Esto no es un defecto en sí de los *scripts*, sino del módulo de procesado de XML.
- Es posible ver y añadir anotaciones, pero no es posible eliminarlas. Si queremos eliminar anotaciones de la base de datos, tendremos que recurrir a un editor de XML externo.
- Es inflexible. La única manera que tenemos de consultar las anotaciones es saber la dirección IP, subred o tipo de alerta a la que se refiere, no hay forma de listar el conjunto de todas las anotaciones.

Sistema de almacenamiento e informe de incidencias Una posibilidad más avanzada de integración con bases de datos, en la misma línea que la anterior, es usar el sistema SISR (*SnortSnarf Incidence Storage and Reporting*), un conjunto de *scripts* que realizan funciones avanzadas. Estos *scripts* utilizan en tiempo de ejecución un fichero de configuración que hemos de indicar al invocar SnortSnarf.

Adicionalmente a los módulos de XML para Perl, SISR requiere el uso de los módulos de Sendmail y HFPM (*HTML Form Processing Modules*), así como del *script* pipeline.pl (el cual debe ser editado para configurar el acceso a los módulos correctamente).

La invocación para SnortSnarf incluyendo soporte para SISR es la siguiente:

```
snortsnarf.pl -d /var/www/html/snort/ -sisr sisr.config \
   /var/log/snort/alert /var/log/snort/portscan.log
```

SISR utiliza también una serie de ficheros de configuración y para almacenar datos:

- Base de datos de conjuntos de eventos (*labeled set database*). XML.
- Base de datos de incidentes (*incident database*). XML.
- Fichero de nombre de los conjuntos de etiquetas. Opcional. Tanto este como los dos anteriores pueden crearse automáticamente con la utilidad setup_sisrdb_dir.pl.
- Base de datos de anotaciones (*annotations database*). Opcional. Permite incluir anotaciones asociadas a una incidencia. Puede ser la misma base de datos que se utiliza para anotaciones sobre direcciones IP, subredes y alertas mencionada en el apartado anterior.
- Plantillas de envío de email.

La gestión de incidentes se hace a través de enlaces proporcionados en cada página con alertas sobre una dirección IP. Con ellos podemos crear conjuntos de eventos, cada uno de ellos consistente en un conjunto de alertas. Podemos crear también incidentes (asociados a un conjunto de eventos), que constan de un conjunto de informaciones en una serie de campos. El número y significado de estos campos es configurable (por ejemplo, dirección IP origen, dirección IP destino, puertos, protocolo, etc.), así como sus valores por defecto en función de los valores de las alertas del conjunto de eventos.

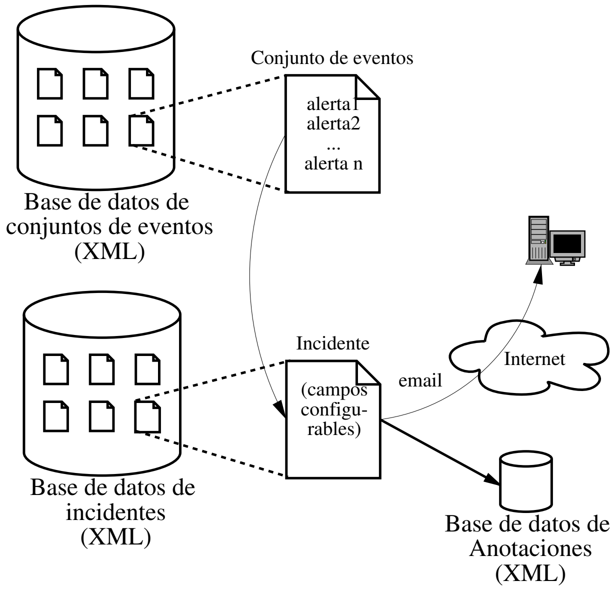


Figura 4.7: Gestión de incidentes con SISR.

Una importante observación es que cuando pedimos la creación de un nuevo conjunto de eventos, se realiza de nuevo la búsqueda sobre los ficheros de entrada (con la opción de aplicar los mismos filtros que se usaron al llamar a snortsnarf.pl —prioridad, redes origen y destino— o no). Esto tiene una ventaja y dos inconvenientes: la ventaja es que si los ficheros de alerta han sido modificados por Snort posteriormente a la creación del informe la obsolescencia de dicho informe HTML no nos afectará. Los inconvenientes son que es necesario que los ficheros de alertas estén situados en un lugar visible para el servidor web y que, puesto que hay que realizar una búsqueda sobre unos ficheros que pueden tener un gran tamaño, existirá un retardo en la creación de conjuntos de eventos[23].

[23]Realmente, la posibilidad de realizar una consulta de los ficheros de alerta actuales puede incorporarse directamente sin necesidad de usar SISR, mediante el *script* text4sel.pl, con la

También tenemos la opción de listar todos los conjuntos de eventos y todos los incidentes, con lo que podemos acceder cómodamente a ellos.

Una vez creado un incidente, opcionalmente podemos realizar una gestión más detallada del mismo. Podemos incluir anotaciones (por ejemplo, para indicar pasos en el proceso de resolución de ese incidente) y enviar emails para informar del mismo (siempre que dispongamos de un servidor de SMTP instalado). Estos emails están basados en plantillas, que facilitan su edición.

Existen limitaciones en este sistema. A parte de compartir los problemas de la base de datos de anotaciones (no se pueden utilizar caracteres especiales, no se pueden borrar registros de las bases de datos) SISR no hace comprobación de si ya existe una entrada igual a la que se pretende introducir, con lo que, si el usuario no tiene precaución, puede haber alertas duplicadas en los conjuntos de eventos.

Otras posibilidades Por último, comentar que a parte de lo indicado, existen opciones para configurar el aspecto más estético de los informes (color, número de entradas por página, etc.), pero que no detallaremos aquí.

4.3.6.2 Guardian

La herramienta Guardian está pensada para integrar las funciones de notificación de alertas de Snort con las de defensa perimetral de un cortafuegos. Guardian es un proceso que se ejecuta en segundo plano, interceptando las alertas que Snort produce en sus ficheros de salida. Cuando se produce una de estas alertas, se comunica con el cortafuegos para bloquear la dirección IP que la ha originado. Podemos configurar que el bloqueo sea temporal (de esa manera, disminuimos el problema que sería bloquear direcciones que en realidad son falsos positivos).

En realidad, lo que hace Guardian cuando tiene que bloquear o desbloquear una dirección IP es invocar un *script* externo (block.sh o unblock.sh), respectivamente. Se proporciona un juego de scripts de este tipo con la distribución para trabajar con los cortafuegos más habituales en entorno GNU/Linux (ipchains e iptables).

El principal problema de Guardian es que la existencia de falsos positivos puede llevar a bloquear direcciones que no son de atacantes. Para evitarlo, podríamos pensar en un sistema basado en las prioridades de las reglas de Snort (es decir, que solo aquellas reglas de un nivel de peligrosidad alto bloqueen). Sin embargo, Guardian no permite realizar esto, al menos directamente.

---

inclusión de -cgidir.

Indirectamente, podemos utilizar subtipos de acciones de reglas para dividir las alertas en distintos ficheros. Por ejemplo, normal_alert asociadas al fichero habitual alert y red_alert, asociadas al fichero red_alert. Al arrancar Guardian, lo configuramos para que intercepte alertas al fichero red_alert. Obviamente, tendremos muy pocas reglas de tipo red_alert, solo para detectar ataques realmente peligrosos.

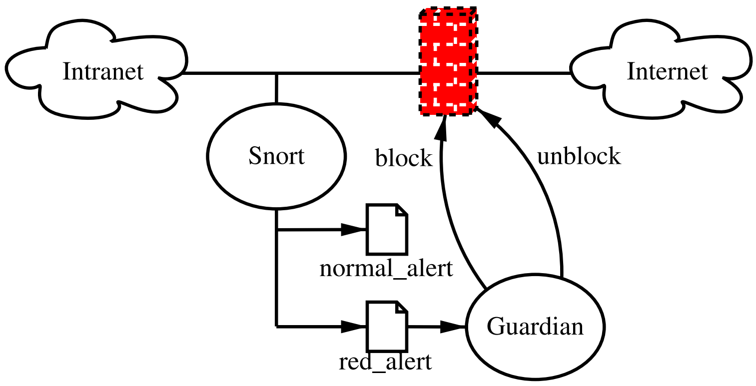


Figura 4.8: Configuración típica de Snort con Guardian.

A pesar de que Guardian está pensado para trabajar con cortafuegos, el hecho de que utilice *scripts* externos da mucha flexibilidad y lo utilizaremos como base para el diseño de las funciones de respuesta activa de la plataforma (apartado 4.4.4).

4.4 Arquitectura

En el presente apartado definiremos que arquitectura utilizaremos en la plataforma MIRA para cumplir los objetivos de funciones de seguridad especificados en el apartado 4.2.3. Realizaremos el diseño de forma progresiva, analizando primero alternativas sencillas pero que presentan problemas, hasta llegar a la solución definitiva.

4.4.1 Diseño basado en la plataforma actual

La solución más sencilla en la que podemos pensar es realizar una modificación en la configuración de la plataforma existente, incorporando un conjunto de patrones extraídos del conjunto de reglas de Snort. No es necesario realizar ningún cambio en los procesos actuales ni en las interfaces de ficheros que utilizan para comunicare entre ellos, que siguen funcionando como se muestra en la figura 2.18. El registro de paquetes sospechosos que poseyeran estos patrones podría realizarse mediante el módulo de registro de flujos (desarrollado como parte del proyecto MIRA en [Lago 00]).

Esta es la solución más sencilla de las que se van a exponer. La arquitectura no sufre grandes modificaciones: los procesos existentes y las interfaces de ficheros que utilizan para comunicarse permanecen inalterados. La implementación consistirá tan solo en el desarrollo de una herramienta que permita la traducción automática de reglas Snort a patrones para *anaflow*, que se agruparían bajo un nuevo síntoma (por ejemplo, SECURITY) o varios síntomas (por ejemplo, con un nombre basado en el fichero del que provienen, como FTP-RULES, DNS-RULES, . . .) en rule.cfg.

Sin embargo, esta alternativa plantea una serie de problemas que la hacen inviable.

- Diferencia en la semántica de las reglas. El principal problema es que las semánticas de reglas de Snort y MIRA son muy diferentes. En concreto las reglas con las que se configura *anaflow* permiten tan solo la detección de patrones, mientras que las de Snort son mucho más complejas y ricas en opciones. Aunque algunas de ellas no son importantes (como sid o rev), existen opciones que no tienen equivalencia en MIRA:
  - ttl, tos, id, ipoption, fragbits, ip_proto, flags, seq, ack, itype, icode, icmp_id, icmp_seq. El proceso de análisis solo inspecciona contenidos, nunca cabeceras IP, TCP, ICMP o RPC.
  - uricontent. Similarmente, requiere inteligencia en el proceso de análisis para saber que partes del paquete se corresponden a la URI.
  - dsize. En *anaflow* se realizan algunas comprobaciones del tamaño del paquete, pero se encuentran "cableadas" en el código y no pueden ser configuradas en las reglas.
  - content. Plantean problemas la negación y los modificadores de extensión de contenido a inspeccionar (offset y depth). Si bien *anaflow* incorpora alguna funcionalidad en este sentido[24], es muy poco flexible.
  - Tampoco tenemos una manera de acceder a la dirección IP o a los puertos del nivel de transporte durante el análisis[25].

[24] Como el $\wedge$, que sería equivalente a offset:0 y depth:longitud del patrón, o el MAXLEN, que es equivalente a un depth global aplicable a todos los patrones (descritos en 2.3.1.2).

[25] Existe la posibilidad de utilizar estos datos para realizar una clasificación directa (fichero de clasificación de direcciones address.cfg). Sin embargo, no nos vale para este propósito, ya que si se realiza dicha clasificación directa no se realiza evaluación de las reglas.

Podríamos considerar la posibilidad de realizar una traducción de reglas que ignorara aquellas opciones que no tienen equivalencia en la plataforma MIRA. Sin embargo, cabe preguntarse que eficiencia tendrán las firmas de ataque así mutiladas, ya que es muy probable que gran parte no resulten útiles.

- Inspección sin estados. La detección de ataques ha de basarse únicamente en la inspección del paquete analizado, no existe "memoria" sobre que paquetes vinieron antes. Debido a ello, no es posible detectar ataques que requieran el mantenimiento de un estado (como realizan los preprocesadores de Snort). Por ejemplo, no es posible detectar una exploración de puertos.

- Categoría de tráfico. A parte de los aspectos de configuración de *anaflow* también hemos de considerar que hacemos en *vflow* con el tráfico en el que se hayan detectado síntomas de seguridad. Podríamos pensar en utilizar una nueva categoría de tráfico (por ejemplo, SEG). Sin embargo, *vflow* no está preparado para ello[26] y, en todo caso, considerar el tráfico de seguridad[27] dentro del algoritmo de clasificación por síntomas ponderados no parece tener mucho sentido.

  Esto es debido a que el tráfico de seguridad será escaso en comparación con el resto de categorías y debido a su importancia, debería ser tratado de forma separada (no podemos arriesgarnos a pasar por alto una alerta en un flujo por el hecho de que se detectaran más síntomas de otra categoría). Una posibilidad sería anteponer un proceso a *vflow* que filtrara el tráfico de seguridad y lo dirigiera a otro sitio. Sin embargo, esto plantea un cambio en la arquitectura de la plataforma, algo que intentamos evitar con esta solución.

- Patrones binarios. Puesto que muchos de los patrones que utilizan las reglas de Snort para detectar ataques son binarios, deberíamos de incorporar funciones en *anaflow* para el reconocimiento de los mismos, algo que la versión original, fuertemente orientada a literales de texto, no permite[28].

---

[26] El proceso asume que las categorías de tráfico son las cuatro prefijadas ACA, LUD, COM e IND, además de que el algoritmo de clasificación está prefijado. No obstante, podría considerarse la posibilidad de flexibilizarlo, tarea que es objeto del apartado 6.4.

[27] Utilizamos el término "tráfico de seguridad" para referirnos al "tráfico en el que se han detectado síntomas de ataques de seguridad" de una forma abreviada.

[28] Realmente, esta objeción no es tan importante como las demás ya que, para incorporar la detección de flujos MP3 que se describe en el capítulo 3, esta funcionalidad tuvo que ser desarrollada de todos modos. Ver apartado 6.2 al respecto.

4.4.2 Diseño basado en Snort como motor de análisis de seguridad

En vista de los problemas planteados en el apartado anterior, parece que la mejor manera de aprovechar toda la potencia de las reglas de Snort es utilizar el propio Snort como motor de análisis de seguridad. En este sentido, podemos plantear la arquitectura que muestra la figura 4.9.

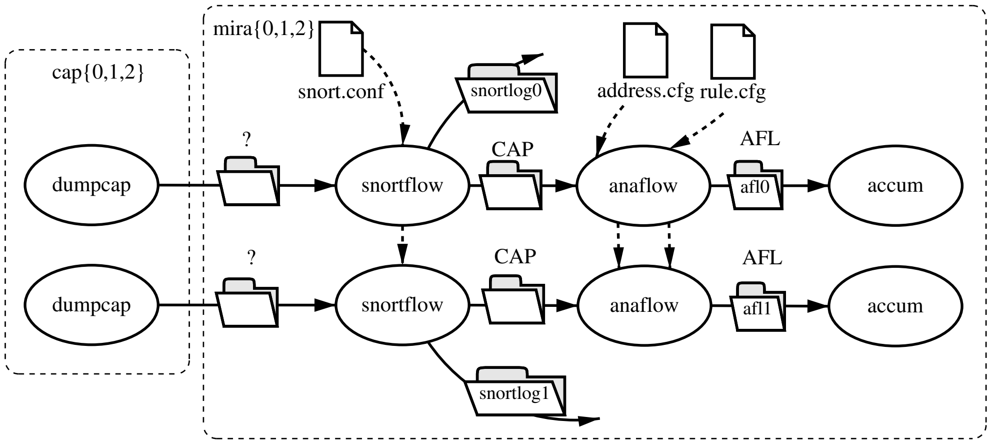


Figura 4.9: Diseño basado en Snort como motor de análisis de seguridad.

El proceso *snortflow* es un recubrimiento de Snort, que se encarga de analizar los paquetes capturados buscando síntomas de seguridad. Este proceso se sitúa en serie con *anaflow*, que realiza la parte del análisis destinada a encontrar síntomas de categorías de tráfico. Ambos procesos se comunicarían usando como interfaz los ficheros CAP.

Los problemas de esta arquitectura están relacionados con las interfaces de entrada y salida que utiliza Snort, que son distintos a los que utiliza la plataforma MIRA.

- Interfaz de entrada. Snort está pensado para funcionar sobre redes de área local y se apoya en la biblioteca de captura de paquetes libpcap. Sin embargo, nuestra interfaz de captura son los fichero CAP. Se nos plantean tres alternativas para solucionar este problema.

  - Realizar la conversión de ficheros CAP a ficheros en el formato de libpcap (de aquí en adelante nos referiremos a dicho formato con PCAP, descrito en el apéndice A.2) mediante un proceso adicional situado entre *dumpcap* y *snortflow* y usar Snort en modo *offline*.

- Modificar el proceso *dumpcap* de las máquinas de captura para que genere ficheros de captura en formato libpcap. Snort se usaría en modo *offline*.
- Realizar la captura con libpcap y usar Snort en modo *online*.

El problema que tiene la última alternativa es que es muy dependiente del escenario de utilización de la plataforma, ya que libpcap corre sobre un número limitado de interfaces de red. En concreto, no existe soporte para ATM[29], con lo cual no es una solución viable para RedIris. De todas formas, aun existiendo teóricamente soporte, habría que revisar la compatibilidad de la librería con el sistema operativo y los controladores del hardware, ya que, salvo para los casos más usuales (por ejemplo, tarjeta Ethernet sobre GNU/Linux) no está garantizado.

A parte de lo referente a libpcap, existe una objeción adicional a tener en cuenta y es que usando Snort en modo *online* estamos fusionando la funcionalidad de módulos de captura y análisis en un mismo proceso, lo cual, desde el punto de vista de la modularidad de la plataforma no es conveniente.

Entre las dos alternativas restantes, en principio resultaría más eficiente modificar directamente *dumpcap*, ya que añadir un nuevo proceso a la cadena de captura y análisis supone una carga adicional de proceso para la plataforma. Sin embargo, *dumpcap* es un software completamente dependiente del escenario de captura, por lo que no es una solución modular.

En conclusión, realizaremos la conversión de CAP a PCAP mediante un proceso adicional, que denominaremos *cap2pcap*. La pérdida de eficiencia puede paliarse aprovechándo la desigual carga de proceso de las distintas máquinas de la plataforma. Recordar (apartado 2.3.2) que las máquinas de análisis están cargadas prácticamente al 100%, mientras que las de captura está cargadas tan solo al 15%. Por tanto, podríamos situar los procesos *cap2pcap* en estas últimas sin disminuir prácticamente nada la eficiencia global del sistema.

Existe un problema adicional (presente en todos los casos) debido a que los ficheros CAP almacenan información de instantes de tiempo de inicio y fin de captura, así como tiempo efectivo de captura durante el periodo (ver apéndice A.1 para una descripción del formato CAP). Si bien la

[29]Esto no es completamente cierto. Existe soporte para ATM cuando se utiliza encapsulado LLC/SNAP, pero no cuando se usa multiplexación basada en circuito virtual ([RFC 1483]), que (como vimos en el apartado 2.4.1) es el caso de RedIris.

información sobre inicio y fin de captura podría inferirse a través de los instantes de captura de la primera y última trama respectivamente, una información que se encuentra en los ficheros PCAP, no existe equivalencia para el tiempo efectivo de captura. La solución a este problema se comentará en la descripción del proceso *cap2pcap* en el apartado 4.5.2.

- Interfaz de salida. Como ya se ha razonado en el apartado anterior, el trafico de seguridad debería ser tratado de forma distinta al tráfico convencional. Observado la figura 4.9, *snortflow* "extrae" el tráfico de seguridad del conjunto de paquetes capturados y deposita el resultado de su análisis en los directorios snortlog0 y snortlog1. El resto del tráfico es el que se pasa al proceso *anaflow* para que lo procese de forma convencional.

  Para la generación de resultados en los directorios snortlog0 y snortlog1, lo más adecuado parece utilizar los métodos estándar de Snort para el registro de alertas y paquetes (descritos en el apartado 4.3.2), garantizando de este modo la compatibilidad con herramientas de post-procesado ya existentes que podemos utilizar. En cualquier caso, la información que se genera localmente en cada una de las máquinas de análisis debería ser integrada para la presentación de información consolidada.

  Una posibilidad interesante sería la utilización de una base de datos centralizada y registrar los incidentes directamente en ella, utilizando el módulo de salida de Snort adecuado (database). De este modo, la integración de los resultados se haría directamente. La consulta de resultados se haría mediante peticiones a esa base de datos, posiblemente integradas dentro del servidor web (por ejemplo, con PHP).

  Además, podrían realizarse procesamientos avanzados de la información allí almacenada (por ejemplo, pedir una lista de las alertas producidas entre dos fechas, con origen en determinada red y destino en tal otra). Sin embargo, puesto que en la definición de la plataforma MIRA no se contempla la utilización de este tipo de bases de datos, descartaremos esta posibilidad.

4.4.3 Diseño basado en Snort como motor de análisis integrado

El planteamiento del apartado anterior, utilizando un proceso para el análisis de seguridad (*snortflow*) y un proceso para el análisis convencional (*anaflow*) de forma separada, plantea algunos problemas.

- Los mismos paquetes son analizados dos veces. Teniendo en cuenta que la tarea más costosa en la plataforma es precisamente el análisis de contenidos, esto no resulta nada conveniente.

- Es necesario que *snortflow* mantenga las interfaces y le pase a *anaflow* un fichero CAP. Por tanto, habría que realizar de nuevo la conversión PCAP a CAP, integrando la información de tiempos que va en un fichero separado. Invertir el orden de los procesos (es decir, primero *anaflow* y luego *snortflow*) podría evitar realizar esta conversión inversa, sin embargo, no resultada adecuado, ya que parece claro que el análisis de seguridad debe ser prioritario sobre el convencional.

Debido a lo anterior, parece más conveniente que el análisis se realice de forma integrada en un único proceso. Esta será la solución definitiva que adoptaremos, desarrollando el programa *snortflow* que no solo recubre a Snort, sino que incorpora la funcionalidad de *anaflow*, al que sustituye. La nueva arquitectura de procesos se muestra en la figura 4.10.

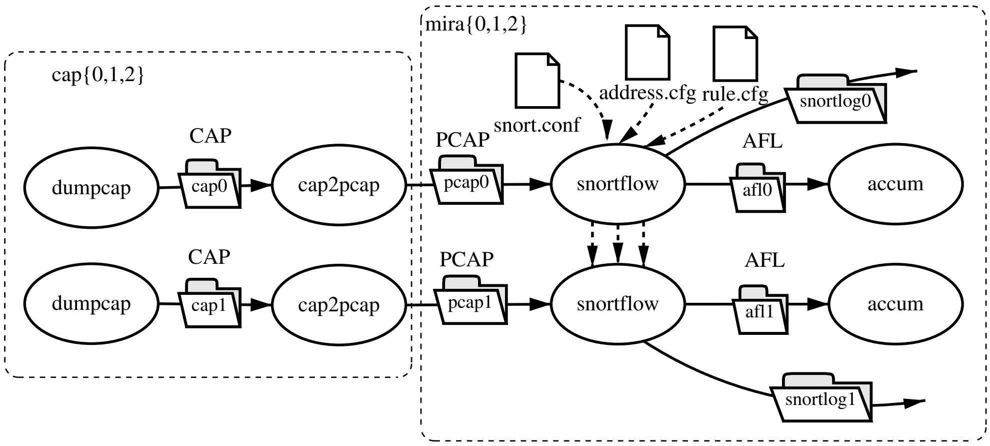


Figura 4.10: Diseño basado en Snort como motor de análisis integrado.

Existen algunas cuestiones a tener en cuenta.

- Es necesario convertir el conjunto de síntomas y direcciones que constituyen la configuración de *anaflow* a reglas de Snort. De todas formas, puesto que la semántica de patrones de MIRA es un subconjunto de la semántica de reglas de Snort, esta conversión no plantea mayor problema.

- La funcionalidad del análisis de Snort no es exactamente igual a la de *anaflow*. En concreto, Snort comprueba si un paquete satisface alguna de sus reglas, pero no comprueba si, una vez disparada una regla, se disparan otras. Este comportamiento, que resulta muy adecuado para la detección de ataques de seguridad (ya que un mismo paquete no va a formar parte de dos ataques simultáneamente), es inapropiado para realizar la detección de síntomas ponderados que requiere posteriormente *vflow* para determinar un veredicto.

- Si un paquete no dispara ninguna regla, no se registra. Esto no es aceptable en nuestra plataforma, ya que este tráfico puede ser correlado más tarde con tráfico sintomático (mediante *biflow*) y, aunque así no sea, es en todo caso tráfico al que *vflow* ha de dar una clasificación.

La solución a estos dos últimos problemas pasa por utilizar Snort como "filtro", dividiendo los paquetes del fichero de captura en distintos grupos:

- Paquetes con síntomas de seguridad.
- Paquetes directamente clasificados en función de su dirección IP y puerto. Tantos grupos como veredictos haya (en la actualidad, cuatro grupos, para ACA, LUD, COM e IND).
- Paquetes en los que se detectó al menos un síntoma convencional.
- Paquetes en los que no se detectó ningún síntoma.

Salvo el grupo de síntomas de seguridad (cuya salida irá directamente a snortlog0 y snortlog1), el resto de grupos serán integrados por *snortflow* para la generación del AFL correspondiente. Será necesario realizar un segundo análisis, al "estilo" *anaflow* sobre el grupo de paquetes en los que se detectó al menos un síntoma convencional, puesto que podrían estar presentes presentes otros síntomas.

Un estudio estadístico ha revelado que, en media y despreciando los paquetes clasificados como incidencias de seguridad, de un fichero de captura típico (1354 paquetes en media), el 1.04% (14 paquetes) se clasifican directamente en función de sus direcciones y puertos, en el 5.46% (74 paquetes) se detectó al menos un síntoma y en el 93.5% (1266 paquetes) restante no se detectó ningún síntoma. Por tanto, hay que observar que el segundo análisis que se realiza es sobre un porcentaje muy reducido de los paquetes sobre los que se realizó el primer análisis y, por tanto, no es significativamente costoso.

4.4.4 Funciones de seguridad activa

En el planteamiento de la seguridad (apartado 4.2.2) se vio como dependiendo del escenario de aplicación de la plataforma MIRA las necesidades de seguridad son distintas. La monitorización pasiva de la red troncal para la generación de información periódica (diaria, semanal o mensual) puede ser un enfoque adecuado, que encaja dentro de la filosofía de la plataforma en lo que respecta a la generación de informes.

Sin embargo, puede ser necesario incorporar algún mecanismo más activo dentro de las funciones de seguridad, pensando sobre todo en la aplicación de la plataforma a redes locales, donde un ataque de seguridad puede comprometer seriamente el servicio ofrecido. El objetivo es realizar una acción determinada cuando una alerta se produce.

La acción concreta a realizar será definida por el administrador de la plataforma, según sea el caso. A continuación se sugieren algunas posibilidades.

1. Defensa perimetral. Una posible acción es modificar la configuración de un cortafuegos introduciendo una nueva regla que bloquee el ataque (por ejemplo, bloqueado la dirección IP origen del ataque). De esta forma, podemos tener un cortafuegos que se adapta dinámicamente a los ataques recibidos.

2. Notificación a un administrador mediante el envío de un correo electrónico. En este mensaje estarían incluidos los detalles de la alerta.

3. Notificación a un administrador mediante un mensaje SMS a un teléfono móvil. Es necesario tener acceso a una estación de envío de mensajes cortos. Este tipo de acción sería la utilizada para alertas realmente críticas, que requieran una acción inmediata del administrador.

4.5 Implementación

4.5.1 Conversión de reglas MIRA a reglas Snort

Para realizar la conversión de reglas se ha preparado un programa, denominado *rm2rs* que, tomando como entrada los ficheros de configuración de *anaflow* (rule.cfg y address.cfg) genera un fichero de salida con las reglas a utilizar con Snort (figura 4.11). La interfaz de llamada puede consultarse en el apéndice B.1.

El fichero generado (snort.mira) tiene cuatro partes.

- Definición de subtipos de reglas. Se definen ocho subtipos[30]:

[30] Realmente, podrían ser más, si admitimos un número variable de categorías de tráfico (mejora descrita en el apartado 6.4), si bien no es este el caso en la versión actual de la plataforma.

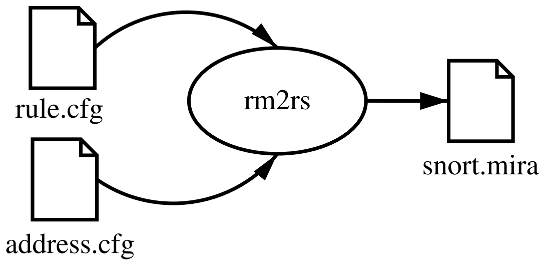


Figura 4.11: Funcionamiento de *rm2rs*.

```
# Ruletypes definition

ruletype red_alert
{
   type alert
   output alert_full: red_alert
}

ruletype mira_alert
{
   type alert
   output log_tcpdump: alert.clog
}

ruletype veredict-LUD
{
   type log
   output log_tcpdump: veredict-LUD.clog
}

ruletype veredict-COM
{
   type log
   output log_tcpdump: veredict-COM.clog
}

ruletype veredict-ACA
{
   type log
   output log_tcpdump: veredict-ACA.clog
}


ruletype veredict-IND
{
   type log
```

```
   output log_tcpdump: veredict-IND.clog
}

ruletype symptom
{
   type log
   output log_tcpdump: symptom.clog
}

ruletype other
{
   type log
   output log_tcpdump: other.clog
}
```

El primer subtipo definido se utilizará para las reglas del sistema de respuesta activa. Si dicho sistema no va ha ser utilizado (como por ejemplo, en la aplicación de de la plataforma MIRA a RedIris), simplemente no tendremos ninguna regla de ese tipo.

Esta configuración nos permitirá dividir el conjunto de paquetes de los ficheros de captura en siete ficheros (con extensión clog para identificarlos), asociados a los distintos grupos mencionados en el apartado 4.4.3 en el primer análisis que realiza *snortflow* haciendo uso de Snort. Estos ficheros están en formato binario (log_tcpdump del apartado 4.3.5) y serán procesados posteriormente por el propio *snortflow*.

El orden de definición de estos subtipos no es arbitrario. Cuando Snort realice el análisis, primero se aplicaran las reglas de alerta (red_alert y mira_alert), las más prioritarias de todas. Posteriormente, se aplicarán las reglas de clasificación directa basadas en direcciones y puertos (veredict-LUD, veredict-COM, veredict-ACA y veredict-IND). Si el paquete aún no ha disparado ninguna regla, se prosigue buscando síntomas en él (symptom). Si todo falla, se le aplica una última regla que realiza una clasificación por defecto (other).

- Reglas de síntomas de tráfico convencional. Extraídas de rule.cfg. Poseen una cabecera symptom completamente ambigua y como opciones, aparte de msg con un texto informativo, tienen una opción content basada en el patrón original. Los patrones iSTR presentan además el modificador nocase.

  El programa convierte correctamente patrones hexadecimales (HSTR) y aquellos que poseen caracteres extraños que son incorrectos en la sintaxis de la regla de Snort (tales como los acentuados), haciendo uso en ambos casos del carácter de escape '|'. Se utiliza el MAXLEN del fichero de configuración MIRA para establecer el depth de todas las reglas.

Los patrones que hacen uso de ∧ indicando que solo deben ser buscados al principio del paquete también se convierten correctamente, utilizando un valor de depth igual a la longitud del patrón (evitando, por tanto, que sea buscado más allá del comienzo del paquete).

Por ejemplo, los siguientes dos patrones MIRA:

```
MAXLEN=500
HTTPS;iSTR;https|;
HTTP;STR;^GET /
```

dan lugar a las siguientes dos reglas Snort:

```
symptom ip any any <> any any (msg: "Rule name HTTPS"; content: "https|7C|"; \
   depth: 500; nocase;)
symptom ip any any <> any any (msg: "Rule name HTTP"; content: "GET /"; depth: 5;)
```

- Reglas de clasificación directa. Extraídas de address.cfg. Se utiliza la cabecera de la regla para especificar dirección y puerto. Como opción se incluye únicamente msg con un texto informativo.

  Por ejemplo, las dos siguientes direcciones:

```
208.202.218.0/27|*|COM|amazon.com|
216.32.74.0/28|*|LUD|yahoo.com|
```

  dan lugar a las siguientes dos reglas Snort:

```
veredict-COM ip any any <> 208.202.218.0/27 any (msg: "Veredict COM";)
veredict-LUD ip any any <> 216.32.74.0/28 any (msg: "Veredict LUD";)
```

- Regla por defecto. Si el paquete no dispara ninguna regla, se le aplica una regla por defecto, para no desecharlo. El efecto es registrarlo en el fichero other.clog.

```
# rule for packets without symptoms and not classified

other ip any any <> any any
```

4.5.2 Desarrollo de *cap2pcap*

El proceso *cap2pcap* es el encargado de realizar la conversión de ficheros CAP a PCAP. Se encuentra situado entre el proceso de captura (*dumpcap*), que proporciona los CAP, y *snortflow* que analiza los ficheros de captura PCAP. Los detalles de formato de ambos ficheros se encuentran en los apéndices A.1 y A.2, respectivamente. La interfaz de llamada se encuentra en B.2.

Para realizar la conversión de un fichero, en primer lugar, se escribe la cabecera global de los ficheros PCAP, utilizando como intervalo de captura el valor 65535 (para capturar paquetes enteros) y como tipo de enlace el 1 (correspondiente a red Ethernet, con lo que estamos simulando un fichero de captura en esa red). Posteriormente, se lee trama a trama del fichero CAP y se escriben los correspondientes paquetes (con sus cabeceras) en el fichero PCAP.

Observar que mientras que los volcados de datos de CAP consisten en las tramas AAL5, los volcados de PCAP consisten directamente en volcados del datagrama IP, en los que la información del nivel subyacente AAL5 se omite. Sin embargo, puesto que estamos "simulando" una red local, este volcado incluirá una cabecera de Ethernet. La utilidad de esta cabecera (con una longitud de 14 bytes) es debida a la necesidad de indicar de alguna forma el VPI/VCI donde se capturó el paquete, campos que no tienen equivalencia directa en el formato PCAP.

Dirección MAC origen 6 bytes. Utilizaremos este campo para indicar el VPI donde se capturó la trama.

Dirección MAC destino 6 bytes. Utilizaremos este campo para indicar el VCI donde se capturó la trama.

Tipo 2 bytes. Indica el tipo de trama. En el caso de IP, se usa el valor 0x0800 (tal como se indica en [RFC 894]).

Por lo demás, la traducción de los campos de la trama CAP a campos de la cabecera de paquete PCAP es inmediata, como muestra la figura 4.12.

Adicionalmente, se realizan algunas comprobaciones de integridad sobre las tramas CAP, para evitar la conversión de datos erróneos que pueda haber introducido *dumpcap*, en concreto que la longitud IP no sea mayor que la longitud AAL5[31].

[31] Inicialmente, también se comprobaba que la longitud AAL5 fuese múltiplo de 48 bytes. Sin embargo, como algunos *dumpcaps* (como el de Ethernet de [Romeral 01]) prescinden de la semántica de trama AAL5, se ha retirado esta comprobación, puesto que puede señalar como erróneas tramas que no lo son.

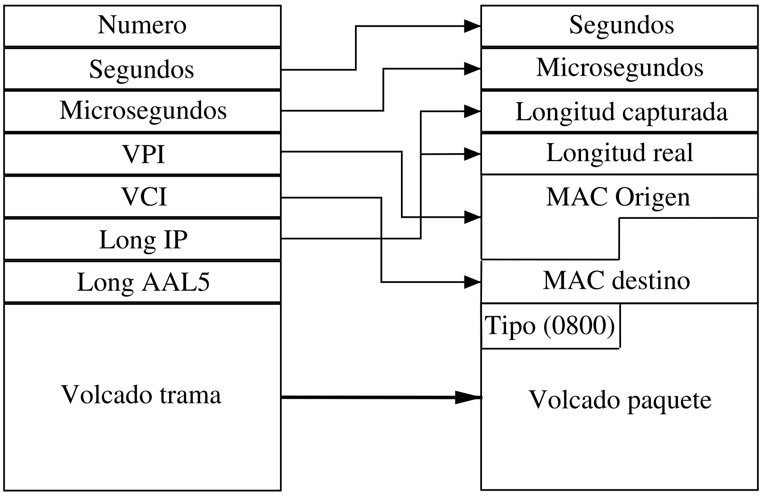


Figura 4.12: Conversión de trama CAP a paquete PCAP.

En el procesado de ficheros CAP existe el problema (ya se señaló en 4.4.2) de la conversión de la información temporal sobre instantes de inicio y fin de captura y tiempo efectivo de la misma, campos que no tienen equivalencia directa en el fichero PCAP. Existen tres posibles soluciones:

- Introducir esta información como un "paquete falso". Sin embargo, es una solución problemática, ya que pueden existir ambigüedades en su interpretación por parte del proceso que use el fichero PCAP (podría interpretarlo como un paquete real).

- Usar "huecos" en el fichero PCAP para almacenar esta información. Los campos VPI y VCI son de 4 bytes en el fichero CAP y los estamos introduciendo en campos MAC de 6 bytes en el PCAP. Por tanto, sobrarían 2 bytes en cada campo (4 en cada paquete) que podríamos utilizar para introducir la información temporal (en total, 24 bytes) en los 6 primeros paquetes del fichero PCAP.

  No obstante, esta solución, a parte de complicar el proceso del fichero PCAP para la extracción de los datos de tiempo, es poco flexible. Por ejemplo, si en una versión futura de la plataforma quisiéramos extender la captura sobre otras interfaces donde los datos de direccionamiento fueran de 6 bytes, no dispondríamos de estos huecos.

- Generar la información de salida en un fichero aparte. De esa forma, *cap2pcap* genera, para cada fichero CAP, dos ficheros de salida: un fichero en formato PCAP donde se encuentra la información de los paquetes capturados que hay en el fichero CAP y un fichero que contiene la información de tiempos, en formato ASCII plano. La sintaxis es la misma que en un AFL (apéndice A.3):

```
init_time=1014818400.552627
final_time=1014821997.247040
cap_time=140.142182
```

Esta fue la solución definitivamente adoptada. El fichero de tiempos (TCAP) puede ser procesado fácilmente por *snortflow*, que tan solo tendrá que anexarlo a los fichero AFL de salida, como se describirá más adelante en 4.5.3.

Por último señalar que, puesto que *cap2pcap* es más rápido procesando que *snortflow*, se incorporan mecanismos de control de flujo como los de *dumpcap* (descritos en 2.3.2).

4.5.3 Desarrollo de *snortflow*

La principal implementación en lo que a funciones de seguridad se refiere ha sido el desarrollo del proceso *snortflow*, que, por su complejidad, se ha planteado en dos etapas.

1. Conseguir un proceso funcionalmente equivalente al de *anaflow*, pero que utilice Snort como motor de análisis.
2. Incorporar las funciones de detección de ataques de seguridad.

La interfaz de llamada de *snortflow* se detalla en el apéndice B.3.

4.5.3.1 Motor de análisis

Lo primero que hace *snortflow* es realizar la conversión de los patrones MIRA a reglas Snort, con la invocación de *rm2rs* (ver apartado 4.5.1). El fichero de configuración así obtenido (snort.mira) será el que se utilizará como configuración. A continuación, el programa realiza el procesado fichero a fichero.

Como se muestra en la figura 4.13, el análisis se realiza en dos etapas. Durante la primera etapa, Snort clasifica los paquetes del fichero de captura en un conjunto de grupos (cada uno asociado a un tipo de regla y a un fichero de salida, con extensión clog). Durante la segunda etapa, se analiza el grupo de paquetes con al menos un síntoma (symptom.clog) buscando todos los síntomas posibles. Simultáneamente a este segundo análisis, se procede con

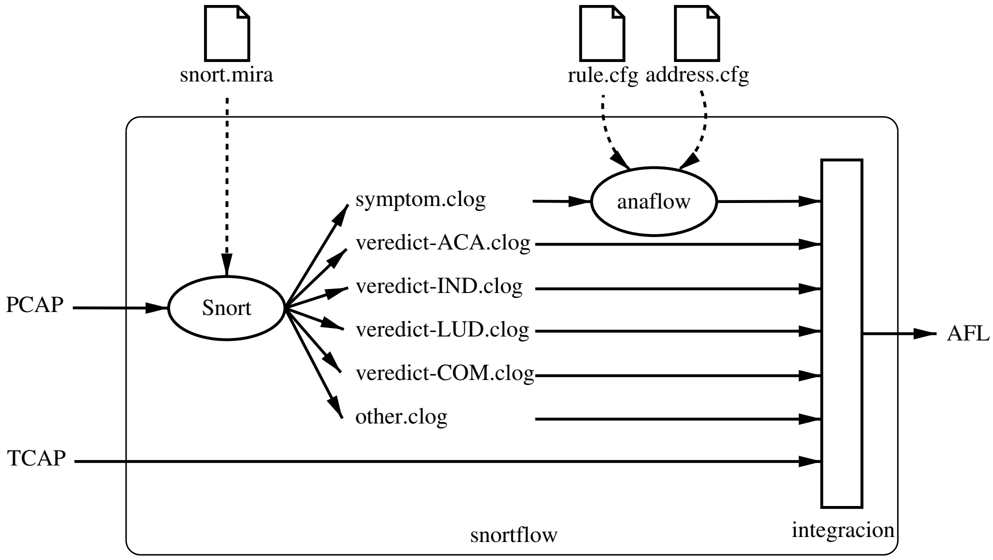


Figura 4.13: *snortflow* como motor de análisis.

la integración de todos los clog para generar el fichero AFL (junto con el fichero de información de tiempo asociado al fichero de captura analizado), que constituye la salida de *snortflow* en esta primera etapa de la implementación.

En concreto, una vez realizado el primer análisis (creados los correspondientes clog, por tanto) se procede de la siguiente manera para la generación de fichero AFL:

1. Procesado de veredict-LUD.clog, veredict-COM.clog, veredict-ACA.clog y veredict-IND.clog. Consiste en leer estos ficheros paquete a paquete y por cada uno de ellos escribir la entrada adecuada del fichero AFL, finalizando la línea con el literal que indica el veredicto (es decir, 'LUD', 'COM', 'ACA' o 'IND' respectivamente).

2. Procesado de symptom.clog. En esta etapa es cuando se realiza el segundo análisis, inspeccionando los contenidos de los paquetes en el fichero. Para cada uno de ellos, se inserta una entrada en el fichero AFL, con la lista de síntomas correspondiente.

3. Procesado de other.clog. Consiste en leer paquete a paquete el fichero e insertar para cada uno de ellos una entrada en el fichero AFL. No hay ni lista de síntomas ni veredicto.

4. Procesado de la información temporal. El fichero AFL finaliza con tres líneas donde se indican los instantes de inicio y finalización de captura,

así como el tiempo de captura efectivo. Esta información se obtiene directamente del fichero TCAP asociado al PCAP que se está analizando, con lo que el procesado consiste simplemente en anexar el TCAP al final del AFL.

Tras finalizar el proceso, los ficheros clog utilizados son eliminados.

El programa *snortflow* incluye el código objeto de *anaflow*, pues es necesario la funcionalidad de este último para la realización del segundo análisis y producir salida AFL. Solo ha sido necesario introducir un cambio, relativo al las funciones de lectura de paquetes, que se implementan en el código objeto frame.o. Hasta ahora, esas funciones (definidas en el fuente frame.c) realizaban lectura de paquetes en formato CAP. Puesto que los clog son escritos por Snort en formato PCAP, ha sido necesario escribir unas nuevas funciones frame_pcap.c y recompilar frame.o. Como puede verse, gracias a la modularidad de los programas, los cambios ha realizar en este aspecto han sido mínimos.

Una vez realizada la implementación, se realizaron pruebas para comprobar su corrección. Para ello se ejecutó *anaflow* y *snortflow* sobre el mismo conjunto de ficheros CAP (previa conversión a PCAP en el segundo caso), obteniéndose la misma salida AFL en ambos casos (salvo por el orden en el que aparecen las entradas dentro del fichero).

También se realizó una evaluación comparativa de la eficiencia de los programas (tiempo que tardan en procesar los mismos CAP) de entrada, obteniendo un resultado similar en ambos casos. Esto podría sorprender en un principio, ya que intuitivamente un proceso que realiza dos veces un análisis (*snortflow*) debería ser más lento que un proceso que solo lo realiza una vez (*anaflow*), demostrándose así la hipótesis del apartado 4.4.3 sobre la poca carga de proceso que añade el segundo análisis. Por otra parte, los tiempos son parecidos porque, en el momento de realizar esta implementación, la búsqueda mejorada de patrones (apartado 6.2) ya había sido incorporada a *anaflow*, con lo que está utilizando el mismo algoritmo que Snort.

4.5.3.2 Funciones de detección de ataques

Una vez implementada la funcionalidad básica de *snortflow*, se pasó a la incorporación de las funciones de detección de ataques. Con el motor de análisis ya construido, la mayor parte de esta implementación consiste en modificar aspectos de la configuración, para la cual utilizaremos el fichero snort.conf (apéndice C.1). Este fichero incluirá el conjunto de reglas convertidas:

```
include snort.mira
```

En primer lugar, hay que construir el conjunto de reglas de seguridad que queremos utilizar. Afortunadamente, la existencia de una base de reglas acompañando a la distribución de Snort nos simplifica mucho el trabajo. Podemos utilizar dicha base de reglas, con una mínima adaptación, que consiste en definir las reglas con los subtipos propios de nuestra plataforma: mira_alert y, para las reglas que queramos que disparen acciones, red_alert. Podemos eliminar las reglas que no nos interesen (la organización modular en ficheros independientes, analizada en 4.3.3.3, es bastante útil para ello) o incluso añadir reglas nuevas específicamente creadas por el administrador.

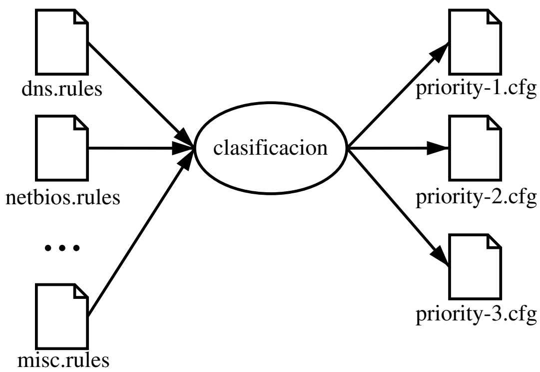


Figura 4.14: Clasificación de las reglas en ficheros por prioridades.

Una posibilidad interesante es agregar este conjunto de reglas y desglosarlas por prioridad, obteniendo tres ficheros: priority-1.rules, priority-2.rules y priority-3.rules (figura 4.14). Posteriormente, podemos incluir tan solo las reglas más prioritarias. De esta forma disminuimos el tamaño de los informes que se generan, reflejando tan solo aquellas alertas realmente peligrosas. En un escenario donde el tráfico es muy elevado, como una red troncal, conviene tener presente este tipo de mecanismos para reducir el conjunto de alertas[32]. Adicionalmente, incluyendo tan solo un subconjunto del total de reglas, la carga de proceso será menor, y la plataforma más eficiente.

En cuanto a la configuración de los preprocesadores, cabe preguntarse hasta que punto puede disminuir su inclusión la eficiencia de *snortflow*. Sin embargo, tras realizar varias pruebas de medida de tiempos, parece ser que su coste es despreciable frente al coste de introducción de reglas. Se ha adoptado la configuración por defecto proporcionada por Snort, que consiste en utilizar los

[32]Interesa que los informes de incidencias no sean excesivamente voluminosos por dos razones: en primer lugar porque ocuparían mucho espacio en el disco de las máquinas de análisis y, en segundo lugar, porque si hay demasiada información se puede colapsar la capacidad de atención del consultor que utiliza estos informes.

preprocesadores frag2, Stream4, http_decode, rpc_decode, bo, telnet_decode y portscan.

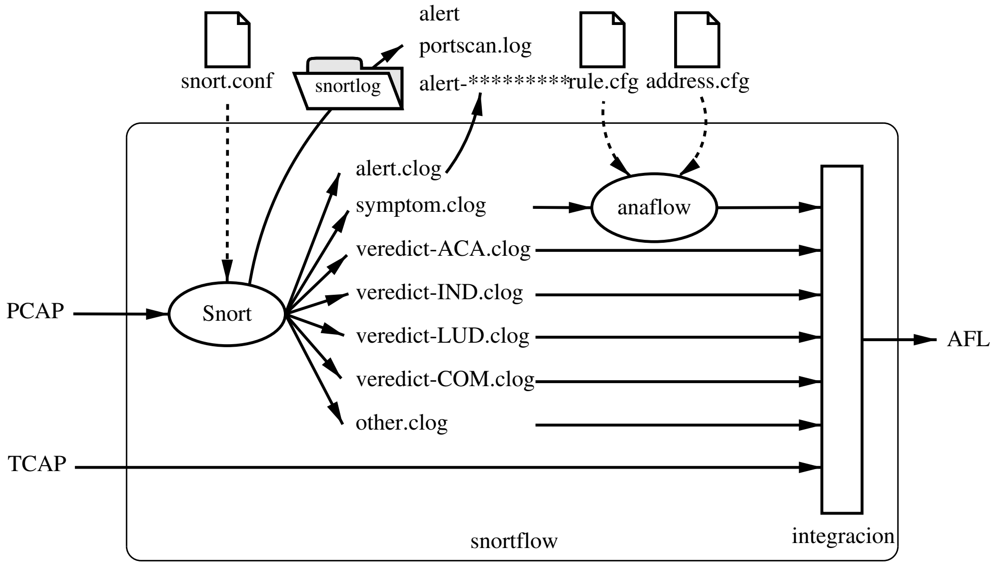


Figura 4.15: Funcionamiento de la versión definitiva de *snortflow*.

En cuanto al procesado de *snortflow*, la figura 4.10 muestra como se añade un paso adicional a la secuencia vista en el apartado anterior, para el procesamiento del fichero alert.clog. Este fichero contiene los paquetes en los que se detectó el ataque de seguridad. Su procesamiento consiste simplemente en un cambio de nombre a alert-*********, donde ********* es un contador decimal de 9 dígitos que se incrementa cada nuevo fichero (el contador es lo suficientemente grande como para no desbordarse durante el tiempo de ejecución de *snortflow*). Observar que si no se hiciera este cambio de nombre, la generación de un nuevo alert.clog sobreescribiría el fichero anterior. Posteriormente estos ficheros serán consolidados (lo cual se describe en el apartado 4.5.5).

En cuanto a las alertas, estas se generan en el fichero estándar alert, que no hemos de procesar con *snortflow*, ya que el propio Snort se encarga de ello automáticamente. También pueden generarse registros de exploraciones de puertos en portscan.log y alertas críticas en red_alert.

4.5.4 Integración con la plataforma

Integramos *cap2pcap* y *snortflow* dentro de los *scripts* de vigilancia y control (mira_ps, mira_ana y control.pl, descritos en 2.3.1.8). En concreto, dentro de mira_ana la activación de estos procesos se realiza con los siguientes comandos[33], utilizando las interfaces de llamada mostradas en el apéndice B.

```
${BIN}/cap2pcap -i ${DAT}/cap0 -o ${DAT}/pcap0 -a 50 &
${BIN}/cap2pcap -i ${DAT}/cap1 -o ${DAT}/pcap1 -a 50 &
rm ${DAT}/snortlog0/*.clog
rm ${DAT}/snortlog1/*.clog
${BIN}/snortflow -i ${DAT}/pcap0 -r ${DAT}/afl0 -c ${CFG}/rule.cfg \
   -a ${CFG}/address.cfg -l ${DAT}/snortlog0 -s ${CFG}/snortconf &
${BIN}/snortflow -i ${DAT}/pcap1 -r ${DAT}/afl1 -c ${CFG}/rule.cfg \
   -a ${CFG}/address.cfg -l ${DAT}/snortlog1 -s ${CFG}/snortconf &
```

Los *rm* son una medida de precaución, ya que si *snortflow* no finalizó correctamente la última vez, pueden existir ficheros clog intermedios, que confundirían al proceso al arrancar de nuevo.

Los cambios en mira_ps y control.pl son mínimos.

4.5.5 Consolidación del tráfico de seguridad y generación de informes

La consolidación de alertas consiste en recoger y agregar la información de seguridad que almacenan las distintas máquinas analizadoras. Hasta ahora, hemos visto como *snortflow* genera unos ficheros de alerta (alert y portscan.log) y un conjunto de ficheros donde está registrado el tráfico que ha producido esas alertas (alert-********), pero no que es lo que se hace con esos ficheros.

La consolidación de alertas se realiza mediante *scripts* de comandos, cuyo código detallado puede encontrase en el apéndice C.2. La consolidación (realizada por consolidate_logs.sh) consiste en dos subprocesos: uno que se realiza en cada una de las máquinas (y que se realiza por invocación a otro *script*, sum_logs.sh en cada una de las máquinas) con el objeto de agregar toda la información de esa máquina y otro (que se realiza tan solo en una de las máquinas) que es la consolidación en sí. El proceso completo (que describiremos a continuación) se muestra en la figura 4.16.

La parte común que se realiza en cada máquina (sum_logs.sh) consta de los siguientes pasos:

1. Detener procesos *snortflow*. Se realiza mediante el envío de la señal de terminación. Esto se hace para que, mientras se ejecuta el siguiente paso *snortflow* no interfiera con el trabajo del *script* generando nuevas alertas.

[33] ${BIN}, ${DAT} y ${CFG} son variables que indican los directorios raíz donde se encuentran los ejecutables que arrancan los procesos, los datos y los ficheros de configuración, respectivamente.

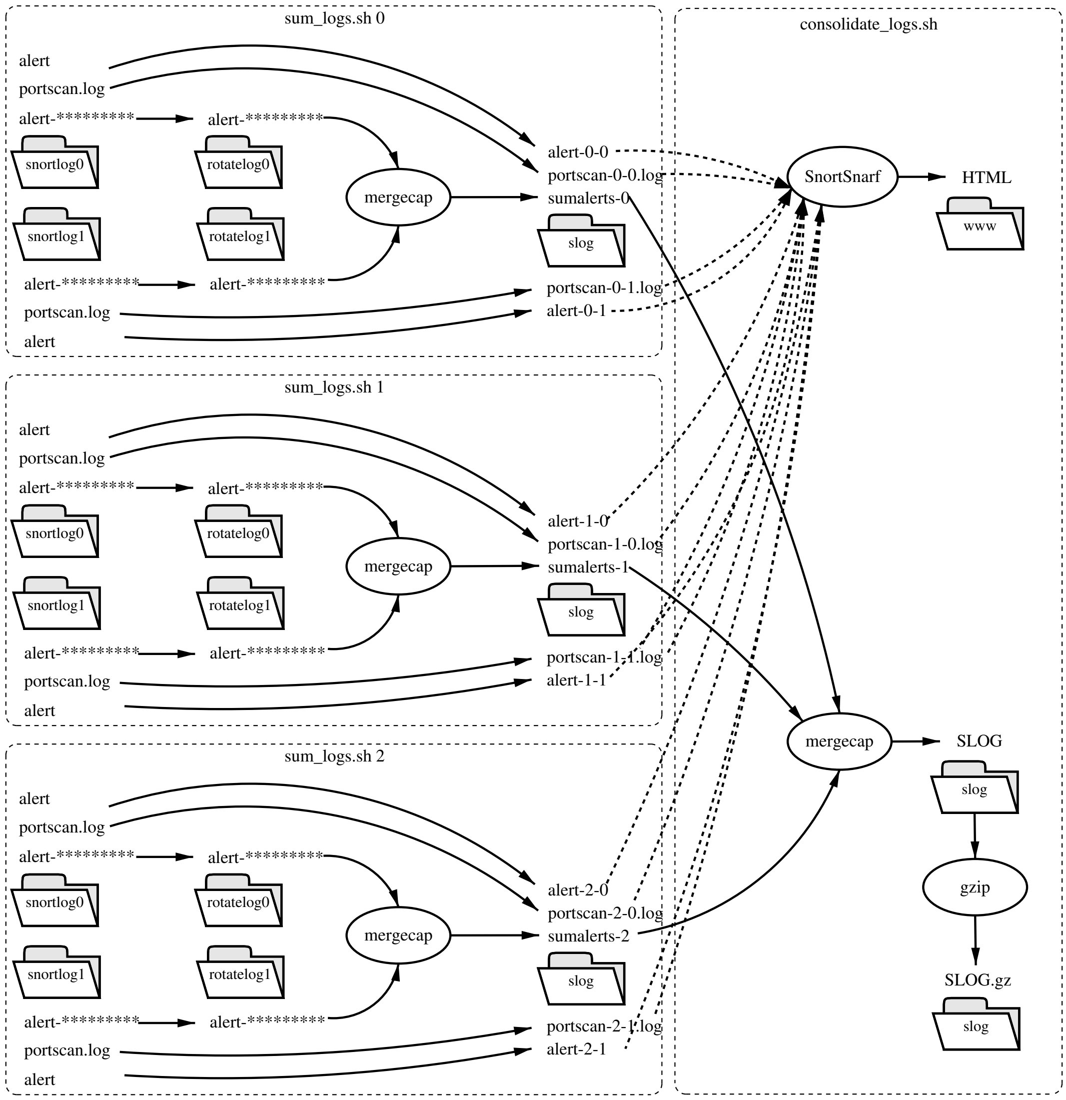


Figura 4.16: Proceso de consolidación de información de seguridad.

2. Mover las alertas y registros de paquete a directorios de trabajo. Es decir, mover el contenido de los directorios snortlog0 y snortlog1 a rotatelog0 y rotatelog1, respectivamente[34].

3. Rearrancar *snortflow*. Observar que detenemos el proceso de análisis el menor tiempo posible y que el resto del proceso del *script* se hará en directorios de trabajo que son independientes de la cadena de análisis. Esto es debido a que el proceso de consolidación puede tardar bastante en realizarse y no conviene detener el proceso de análisis de la plataforma mientras tanto.

4. Agregar el conjunto de paquetes registrados. Todos los ficheros alert-********* de ambos sentidos de captura han de ser unificados en uno solo. Para ello, podemos utilizar la herramienta *mergecap*. El nombre del fichero agregado incluye un identificador de la máquina en la que ha sido generado (por ejemplo, sumalerts-1, generado en la máquina analizadora número 1).

   El fichero agregado se deposita en slog, que es un directorio compartido NFS en mira0 al que pueden acceder todas las máquinas analizadoras.

5. Mover los ficheros de alertas (alert y portscan.log) al directorio compartido. Al moverlos, se renombran, indicando la máquina y sentido de captura (por ejemplo, alert-1-0 es el fichero de alertas de la máquina 1 en el sentido 0 —salida—, portscan-2-1.log es el fichero de registro de escaneo de puertos de la máquina 2 en el sentido 1 —entrada—).

6. Limpiar el contenido de los directorios de trabajo rotatelog0 y rotatelog1.

En resumen, al termino de este procesado tendremos los siguientes ficheros en slog[35]:

mira0	mira1	mira2
sumalerts-0	sumalerts-1	sumalerts-2
alert-0-0	alert-1-0	alert-2-0
alert-0-1	alert-1-1	alert-2-1
portscan-0-0.log	portscan-1-0.log	portscan-2-0.log
portscan-0-1.log	portscan-1-1.log	portscan 2-1.log

[34] Esta reubicación de ficheros no incluye el fichero red_alert, cuyo tratamiento especial se describe en el apartado 4.5.6.

[35] Asumiendo que tenemos tres máquinas analizadoras (apartado 2.4.1). Puede ser distinto en otro escenario de aplicación, pero en todo caso, la solución presentada aquí es fácilmente escalable.

A continuación, se completa la consolidación (consolidate_logs.sh):

1. Consolidar las alertas de las tres máquinas. Se vuelve a utilizar *mergecap* para agregar los ficheros sumalerts-0, sumalerts-1 y sumalerts-2. Este es el fichero definitivo y para nombrarlo utilizaremos la sintaxis habitual de MIRA, con un prefijo (SLOG en este caso) y la marca de tiempos de 12 dígitos habitual correspondiente al instante en el que se ha realizado la consolidación.

2. Se comprime el fichero anterior, para reducir la ocupación de disco duro.

3. Crear informe HTML. Se genera un informe mediante la herramienta SnortSnarf, utilizando como entrada los 12 ficheros de alerta correspondientes.

4. Borrar los ficheros intermedios generados durante la consolidación.

Vamos a analizar más en detalle aspectos relacionados con esta implementación.

4.5.5.1 Periodo de consolidación

El *script* consolidate_logs.sh está diseñado para ser invocado periódicamente. Esto puede conseguirse mediante la utilización de *cron*, que es un demonio existente en entornos UNIX que nos permite arrancar un proceso cada cierto tiempo.

El periodo de invocación debería ser determinado cuidadosamente teniendo en cuenta las características del escenario de aplicación. Por una parte, no puede ser demasiado corto, ya que realizar la consolidación lleva cierto tiempo y, con un periodo muy corto, la capacidad de proceso estaría más tiempo destinada a la generación de informes que al propio análisis de los contenidos.

En escenarios donde haya poca intensidad de tráfico (por ejemplo, una red local poco cargada), el periodo de consolidación puede ser relativamente grande (semanal o mensual). En escenarios donde el tráfico sea muy intenso (típicamente, una red troncal, como RedIris), se recomiendan periodos de consolidación diarios o incluso horarios.

4.5.5.2 Herramienta *mergecap*

La utilidad *mergecap*, que se distribuye como herramienta en las distribuciones más recientes de [Ethereal] (que será necesario instalar en las máquinas de análisis), es un programa en línea de comandos que permite agregar un conjunto de ficheros en formato PCAP, como son los que nosotros manejamos para el registro de paquetes.

Toma como argumentos un nombre de fichero de salida (parámetro -w) y un conjunto de ficheros de entrada, realizando la agregación de todos estos últimos en un único fichero, con el nombre especificado.

La herramienta realiza una agregación que respeta inteligentemente las marcas de tiempo. Los paquetes se encuentran ordenados por su valor de marca de tiempo dentro del fichero agregado, no se limita simplemente a anexar un fichero detrás de otro.

Al invocar el programa podemos utilizar opciones que configuran su funcionamiento y que se encuentran detalladas en su página de manual.

4.5.5.3 Generación de informes

Se ha elegido como mecanismo de reporte de informes la utilización de páginas HTML que serán integradas dentro del web dinámico de la plataforma junto con el resto de informes sobre caracterización de tráfico, para su acceso remoto por parte de los consultores (descrito en el apartado 2.3.3).

Dentro del servidor, estos informes colgarán del directorio `snort`, dentro del cual tendremos un subdirectorio para cada uno de ellos, cuyo nombre será una marca de tiempo del momento en el que se generó. Por ejemplo, en la URL http://localhost.localdomain/snort/31-12-01-23:59:50/ contiene el informe HTML generado el día 31 de diciembre de 2001, a las 23:59:40 horas.

Para la generación de estos informes se utiliza la herramienta SnortSnarf. En el apartado 4.3.6.1 vimos como esta herramienta admite múltiples posibilidades de utilización. Es el administrador de la plataforma quien deberá decidir cuales son las opciones más convenientes, teniendo en cuenta el escenario de aplicación.

En todo caso, se recomienda utilizar al menos una ordenación de las alertas basada en la prioridad, de forma que el consultor puede determinar que alertas producidas son más peligrosas y atenderlas primero.

No se recomienda que el acceso a los registros de paquetes se haga desde la misma página web, ya que habría que tener estos registros descomprimidos y en formato ASCII, ocupando un gran volumen de disco duro. Si la información arrojada por los informes no es suficiente para el consultor, este siempre podrá ir a los SLOG, como se describe en el siguiente apartado.

4.5.5.4 Gestión posterior de los ficheros SLOG

A parte de los informes, la consolidación de registros arroja como resultados ficheros SLOG. En cada uno de estos ficheros se encuentra almacenado el conjunto de paquetes que dispararon reglas de seguridad comprendidos entre la marca de tiempo del fichero y la marca del tiempo del fichero inmediatamente anterior.

Estos paquetes permiten investigar más a fondo la forma en la que se han realizado los ataques de seguridad. Para inspeccionarlos, pueden utilizarse herramientas gráficas como [Ethereal], potente y versátil, o incluso el propio Snort, que puede generar el árbol de ficheros de registros ASCII (figura 4.3) a partir de un fichero binario PCAP.

Sin embargo, puesto que estos son ficheros voluminosos cuyo número puede crecer con relativa rapidez, conviene definir una política de borrado que evite el colapso del espacio de disco en la máquina de consolidación. Para ello podemos utilizar el programa *coll* (apartado 2.3.1.8) ya utilizado para realizar funciones similares (como borrado de los VFL), configurándolo para que elimine aquellos ficheros SLOG con una antigüedad mayor que un tiempo determinado[36].

4.5.6 Funciones de respuesta activa

En la implementación considerada en el apartado 4.5.5 puede parecer que se ha descuidado el procesado de los registro de alerta red_alert. Tal omisión es deliberada, ya que estas van a ser alertas de un elevado nivel de peligrosidad, por lo que se producirán en un número demasiado escaso como para justificar su integración y, más importante, requieren una respuesta inmediata que no puede esperar a que se realice el proceso de consolidación.

Así pues, existirá un fichero red_alert en los directorios snortlog0 y snortlog1. Para cada uno de ellos establecemos un proceso Guardian, el cual reaccionará cada vez que Snort escriba una alerta en uno de ellos. Esta reacción consiste en la invocación del *script* block.sh, que será el que realice la acción oportuna. Guardian realiza la llamada a block.sh usando como argumento la dirección origen del ataque.

La gestión de los registros red_alert ha de realizarse manualmente, lo cual no es demasiado problema, si tenemos en cuenta que no se producirán muchas de estas alertas.

Obsérvese que para las reglas red_alert no se produce registro del paquete que ha generado la alerta. Tal funcionalidad sería deseable, aunque obligaría a algunos cambios en la configuración de *snortflow*. No obstante, podría considerarse como una mejora posible a las funciones de seguridad de la plataforma.

[36] Por ejemplo, si el periodo de consolidación es de una hora, pueden borrarle los SLOG con más de una semana de antigüedad, aunque esto depende de múltiples factores: disponibilidad de espacio de disco, necesidad de registrar tráfico sospechoso para realizar acciones de denuncia, ...

4.6 Conclusiones

Una vez expuesto el proceso de análisis, diseño e implementación que ha conducido a la incorporación de funciones de seguridad en la plataforma MIRA, se señalarán a continuación algunas conclusiones de interés.

Lo primero que hay que destacar es que el uso de desarrollos externos puede solucionar problemas que, si hubiera que tratar directamente partiendo de cero, serían muy complejos y costosos de resolver. En concreto, el uso de la herramienta de seguridad Snort ha permitido abordar el complejo problema de la seguridad sin tener que realizar una laboriosa tarea de investigación sobre el tema.

Más aún, no solo soluciona el problema original presente sino también la evolución futura, ya que Snort es un proyecto activo que continuamente está produciendo nuevas versiones y con ellas las reglas que detectan las firmas de ataques de nueva aparición. En el campo de la seguridad lógica, como ya se ha señalado anteriormente, estar al día es de vital importancia.

En segundo lugar, queremos destacar la *integración* de las funciones de seguridad desarrolladas con el resto de la plataforma de análisis de tráfico. El proyecto MIRA aporta este aspecto novedoso, ya que si bien es posible encontrar en el mercado herramientas especializadas en seguridad y herramientas especializadas en análisis de tráfico, es difícil encontrar una que integre ambas funcionalidades, obligando a la utilización de plataformas de ambos tipos en paralelo y perdiendo el valor añadido de la integración.

Así mismo, se quiere destacar la *flexibilidad*, ya que el sistema implementado, que permite combinar informes de tráfico de seguridad (considerados a medio y largo plazo) con notificaciones inmediatas (que requieran una respuesta a corto plazo), es adaptable a todo tipo de situaciones.

No obstante de las dos características anteriores, antes de plantearse la utilización de la plataforma MIRA como herramienta de monitorización de seguridad, habría que considerar las posibles alternativas. La verdadera utilidad de MIRA es la de integrar un conjunto de funciones de análisis de tráfico, de las cuales las relacionadas con la seguridad son solo una parte. Por tanto, si lo que se desea es tan solo la monitorización de seguridad, son más óptimas las soluciones basadas en Snort (especialmente, si nos encontramos en un escenario de red local).

Por último, señalar algunos puntos débiles del desarrollo aquí realizado. A pesar de que se ha implementado una solución completa al problema de la seguridad, no se han realizado pruebas exhaustivas de la misma. Resultaría conveniente probar la estabilidad del sistema a largo plazo[37].

[37] Por ejemplo, comprobar que nivel de ocupación de disco duro suponen los registros de

Así mismo, también sería conveniente realizar un estudio del impacto que tienen las nuevas funciones sobre la eficiencia de la herramienta. De esa forma podría determinarse cual es el número óptimo de firmas de seguridad a utilizar, en que medida influye el uso de preprocesadores, como afectan los falsos positivos, ... No tendría sentido la integración de funciones de seguridad en la plataforma si la carga que supone imposibilita el análisis de tráfico convencional.

Por último, relacionado con los aspectos de eficiencia, señalar que la forma en la que ha sido realizado el acoplo entre Snort y la plataforma no es del todo perfecta, ya que se realiza a través del sistema operativo (*snortflow* invocan Snort externamente con una llamada a la *shell*). Se obtendrían mejores resultados si estuvieran integrados dentro de un mismo proceso. Esta solución exigiría un estudio del código fuente de Snort, posibilidad que se desestimó en el presente desarrollo debido al elevado grado de complejidad, pero que podría constituir una línea de trabajo futuro con el objetivo de mejorar la integración de las funciones de seguridad en la plataforma (y como tal se señala en el apartado 7.2.5).

seguridad, ya que es concebible que estos puedan desbordar el espacio de almacenamiento.

Capítulo 5

Estadísticas convencionales

5.1 Introducción

Si bien el punto fuerte de la plataforma MIRA es la clasificación del tráfico a través de la inspección de contenidos de los paquetes, para ciertos tipos de análisis resultan también útiles las estadísticas que se basan tan solo en los campos que aparecen en las cabeceras de los mismos. Por ejemplo, tal puede ser el caso de estudios de utilización de puertos de transporte.

La funcionalidad a conseguir sería similar a la de un analizador de protocolos convencional y en ese sentido, empleamos el término "estadísticas convencionales". Es de destacar que no se trata de imponer este tipo de estadísticas a las de inspección de contenidos, ni viceversa. La intención es contar con una información complementaria, que, para determinados tipos de análisis de tráfico puede resultar útil para el consultor.

Mediante la incorporación de esta funcionalidad se consigue dar mayor versatilidad a la plataforma MIRA. De esta forma, para la obtención de estadísticas convencionales no es necesario la utilización de una plataforma de análisis de protocolos en paralelo a la plataforma de inspección de contenidos (solución que, por otra parte, podría mermar las capacidades de eficiencia global y carece del valor añadido de la integración), realizándose ambos análisis de tráfico de forma conjunta.

5.2 Análisis

5.2.1 Criterios de clasificación

Las estadísticas que queremos conseguir han de extraerse examinando exclusivamente cabeceras de los paquetes, nunca contenidos. Los criterios de clasificación para la generación de estadísticas convencionales utilizan exclusivamente dos campos:

- Protocolo a nivel de transporte. Distinguiremos tres casos: TCP, UDP u otro protocolo (es caso más significativo dentro de esta última categoría sería ICMP[1]).
- Puerto de direccionamiento. En el caso TCP y UDP, nos fijaremos en el puerto utilizado (0–65535). En el caso de otros protocolos, puesto que no todos utilizan un espacio de direccionamiento de este tipo, no haremos mayor distinción.

Como puede verse, ambos son campos que aparecen dentro de la cabecera de los paquetes (cabecera IP y de transporte, respectivamente), no hay inspección del contenido en ningún caso.

5.2.2 Tipos de estadísticas convencionales

Por una parte, una de las estadísticas que podemos considerar son aquellas que muestran simplemente la utilización (en bytes cursados) de los distintos puertos (para protocolo TCP y para protocolo UDP). Se trata de una información a *bajo nivel*.

Aunque podríamos considerar únicamente estas estadísticas de bajo nivel, puede no resultar demasiado útil para un consultor que estará más interesado en una información de *alto nivel*. Por tanto, más que utilizar la información de uso de puertos en sí, lo que haremos será basarnos en ella para clasificar por aplicaciones y grupos de aplicación.

Se define una *aplicación de tráfico* como un conjunto de puertos TCP y un conjunto de puertos UDP. Todo el tráfico que usa esos puertos se considera clasificado dentro de esta aplicación. Definimos también *grupo de aplicación*, como una abstracción de mayor nivel que consta de un conjunto de aplicaciones, siendo el tráfico clasificado dentro del grupo como el agregado del tráfico que clasifican las aplicaciones de las que consta. Vamos a clarificarlo con un ejemplo.

Consideremos un grupo de aplicación MAIL. Este grupo consta de tres aplicaciones: SMTP, POP e IMAP. La aplicación SMTP consta del puerto TCP 25, la aplicación POP de los puertos TCP 109 y 110 y la de IMAP de los puertos TCP 143 y 220. En un momento dado tenemos la siguiente distribución de tráfico (en bytes).

[1] Aunque la denominación como "protocolo de transporte" no es estricta en algunos casos (por ejemplo, ICMP y IGMP son subconjuntos de IP, no son protocolos de transporte sino de gestión de red) obviaremos este detalle.

...	
Puerto TCP 25:	5000
Puerto TCP 109:	1000
Puerto TCP 110:	1500
Puerto TCP 143:	2000
Puerto TCP 220:	2300
...	

Como vemos, es una información de muy bajo nivel, que no nos dice mucho. Sin embargo, a partir de ella podemos llegar a la clasificación por aplicaciones:

SMTP:	5000
POP:	2500
IMAP:	4300

Y a partir de esta, a la clasificación del grupo MAIL, con 11800 bytes.

5.2.3 Información sobre aplicaciones y grupos de aplicación

Uno de los problemas que plantea la clasificación basada en aplicaciones y grupos de aplicación es que hemos de encontrar una manera fidedigna de relacionarlos con los puertos y protocolos. Es decir, qué aplicaciones considerar y que puertos asignar a cada una o, dicho de otro modo, que conexión hay entre las clasificaciones a bajo y alto nivel.

Para conseguir esto, es necesario un estudio de las distintas aplicaciones que se emplean en Internet (o al menos de las más utilizadas, pues será en ellas donde consigamos clasificar más tráfico) y de que protocolo y puerto utilizan. Más aún, debido a la rápida evolución de las aplicaciones de red (continuamente se están desarrollando nuevos servicios), es necesario que este estudio sea dinámico, pudiendo considerar el surgimiento de nuevas aplicaciones y grupos de aplicación.

Afortunadamente, existen asociaciones que se dedican precisamente a estudiar estos aspectos y cuya información podemos utilizar. En concreto, para generar las estadísticas convencionales de MIRA hemos utilizado un fichero de clasificación de aplicaciones por protocolo y puerto incluido con la distribución de la herramienta [CoralReef], desarrollada por CAIDA (*Cooperative Association for Internet Data Analysis*).

Sin embargo, podemos considerar la aparición de los siguientes inconvenientes.

- Puertos que no forman parte de ninguna aplicación.

- Puertos que podrían formar parte de varias aplicaciones.

Con respecto al primero, hemos de tener en cuenta que, si bien los servidores utilizan puertos concretos (por ejemplo, puerto 80 para el servicio web), no es así en los clientes, que suelen utilizar un puertos aleatorios de alto valor[2] para abrir la conexión con el servidor.

El agregado de todo ese tráfico de puertos aleatorios origen que usan los clientes, con un volumen significativo en nuestras estadísticas (aunque disperso en un gran número de puertos), no podrá ser clasificado. Sin embargo, normalmente esto no será el objetivo de la estadística, sino el estudio de los servicios de puertos conocidos, por lo que podemos ignorarlo.

Con respecto al inconveniente de que varias aplicaciones compartan un mismo puerto, si dichas aplicaciones pertenecen a un mismo grupo, no es tanto problema. Sin embargo, si pertenecen a grupos distintos, tendremos que tomar una decisión sobre a que aplicación asignamos los puertos comunes, normalmente en base a la aplicación que creamos es más utilizada, con el objetivo de disminuir el porcentaje de tráfico que resulte erróneamente clasificado.

5.3 Objetivos del diseño

Lo que a continuación se va a describir es el módulo de estadísticas convencionales de la plataforma MIRA. Este módulo es un proceso que tomando como entradas los datos de tráfico, genera las estadísticas convencionales (tanto las estadísticas de grupos, de alto nivel, como las de puertos, de bajo nivel).

Una de las premisas de diseño es que el módulo fuera lo menos intrusivo posible con la plataforma ya existente. Es decir, que la arquitectura de procesos mostrada en la figura 2.18 se altere lo menos posibles. El módulo de estadísticas convencionales toma como punto de salida los ficheros VFL que genera el proceso *vflow* en cada una de las máquinas de análisis. Cada uno de estos ficheros contiene información del tráfico acumulado a lo largo de una hora. Constan de una serie de líneas con información sobre cada flujo bidireccional (apéndice A.5), sin embargo desde el punto de vista del módulo de estadísticas convencionales estaremos interesados solo en los siguientes campos:

- VPI/VCI de la Comunidad Autónoma.
- Protocolo de transporte utilizado.
- Puertos local y remoto.

[2]Precisamente porque los puertos bajos (menores de 1024) son los *well-known ports*, reservados para servicios estandarizados.

- Bytes de entrada y salida.

Estos ficheros VFL generados por *vflow* sirven de entrada a los procesos que gestionan la información del web dinámico (apartado 2.3.1.8). Por tanto, la inserción del módulo de estadísticas convencionales se realizará entre *vflow* y estos procesos, utilizando un directorio intermedio `vfl-t`. La modificación se muestra en la figura 5.1.

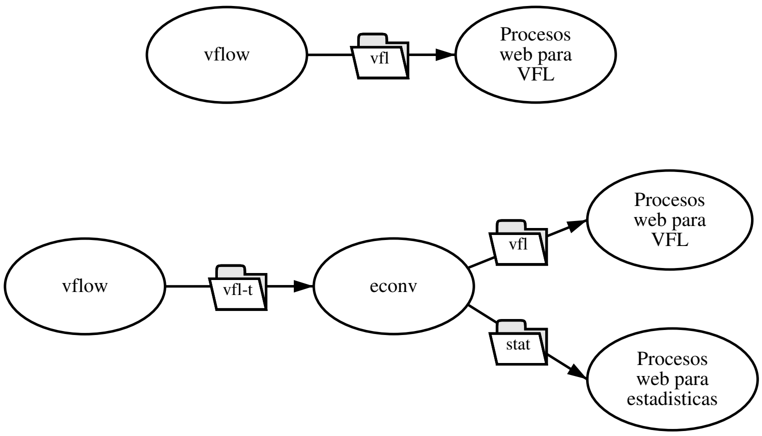


Figura 5.1: Proceso de los VFL convencional (arriba) y con *econv* (abajo).

Los procesos del web para el procesado de VFLs no sufren ningún cambio en su interfaz de entrada. El directorio `stat` contendrá las estadísticas de salida organizadas en un árbol de subdirectorios, que es la interfaz de entrada que necesitan los procesos del web de procesado de las estadísticas.

Los objetivos del diseño se concretan en:

- Extraer los datos brutos sobre uso de puertos presentes en cada fichero VFL, devolviendo como salida este mismo VFL con una clasificación de cada uno de los flujos. No deberá modificarse sustancialmente el formato de ese fichero, para que los procesos del web dinámico no tengan que sufrir modificaciones importantes.

- Clasificar el tráfico de los puertos en aplicaciones y grupos de aplicación.

- Generar los ficheros que contienen las estadísticas convencionales. Se generarán dos tipos de ficheros: por grupos de aplicación (estadísticas de alto nivel) y por uso de puertos TCP y UDP (estadísticas de bajo nivel). Además, puesto que se desea que el consultor pueda acceder a estos datos a través del web dinámico, la ubicación de estos ficheros deberá ser la correcta para que se integre con los procesos de gestión de dicho web.

5.4 Implementación

El módulo de estadísticas convencionales consiste en el proceso *econv*, está diseñado de forma modular y consta de un conjunto de clases que agrupan datos y métodos específicos para realizar distintas acciones, cuyo diagrama UML se muestran en la 5.2. La interfaz de llamada se detalla en el apéndice B.4.

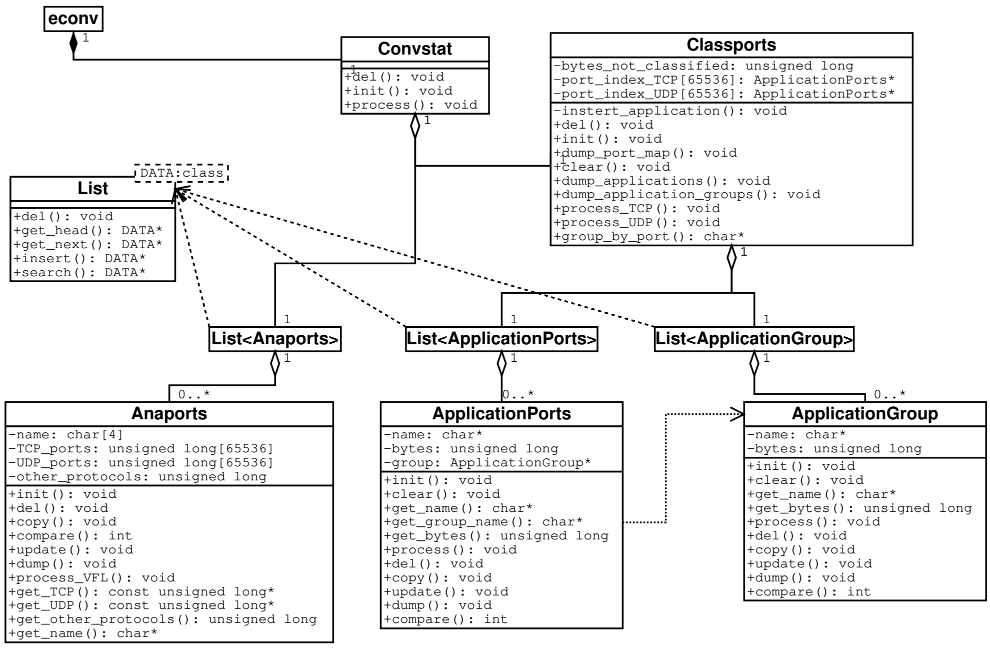


Figura 5.2: Diagrama UML de clases de *econv*.

El programa utiliza para realizar el procesado y generación de estadísticas un objeto *Convstat*, que posee una lista de *Anaports* y un *Classports*.

5.4.1 Inicialización del clasificador

El *Classports* es el encargado de realizar la clasificación. Para construir las estructuras de decisión se utiliza el método *init*, que utiliza como argumento un fichero de configuración donde aparece la relación entre puertos, aplicaciones y grupos de aplicación. Este fichero, mencionado en 5.2.3 es el de [CoralReef].

Durante esta inicialización, para cada entrada del fichero (en una sintaxis específica que *init* entiende) se insertan entradas en el árbol de clasificación (mediante el método privado *insert_application*) en una lista de *ApplicationPorts* y en una de *ApplicationGroup*.

Cada aplicación de la lista (*ApplicationPorts*) está indexada por todos aquellos puertos que utiliza (utilizando los vectores *port_index_TCP* y *port_index_UDP*). A su vez, cada grupo de aplicación de la lista (*ApplicationGroup*) está indexado desde los *ApplicationPorts* que la usan mediante el puntero *group*. Ambas tienen un contador de tráfico (*bytes*) de forma que si tenemos un tráfico asignado a cierto puerto UDP o TCP, resulta inmediato (mediante la indexación) asignarlo a la aplicación y grupo de aplicación correspondiente. El propio *Classports* tiene un atributo (*bytes_not_classified*) para contabilizar el tráfico que no pueda clasificarse en estas listas.

En definitiva, la estructura creada por *init* es la que muestra la figura 5.3.

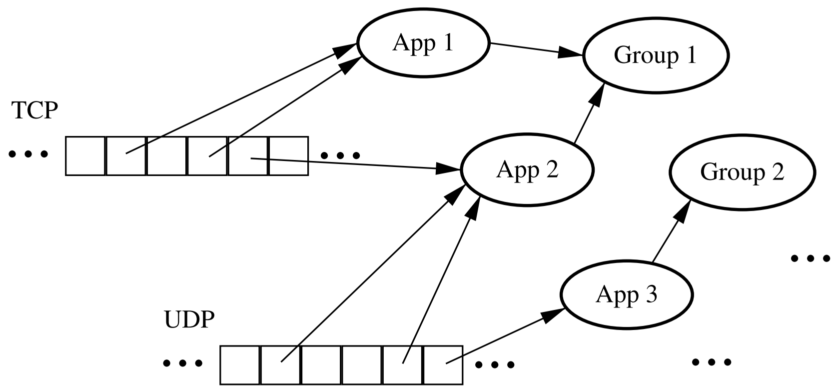


Figura 5.3: Estructura de clasificación de *econv*.

5.4.2 Procesado de VFL

El programa funciona en segundo plano, procesando los ficheros VFL que genera *vflow* y generando por cada uno de ellos otro fichero VFL para los procesos de gestión del web dinámico (básicamente, el mismo VFL de entrada, salvo por la pequeña modificación que se comenta a continuación). Para cada uno de estos ficheros, se invoca *process* de *Convstat*, que realiza los siguientes pasos:

- Reunir información de puertos. Se lee el fichero línea a línea, acumulando el tráfico de los protocolos y puertos. En concreto, se acumula información sobre puertos TCP, puertos UDP y otros protocolos (agregados) en los vectores de contadores *ports_TCP* y *ports_UDP* y en el contador *other_protocols* de objetos *Anaports*. Cada uno de los *Anaports* está asociado a una Comunidad Autónoma (identificada por su VPI/VCI), por lo que con este sistema contabilizamos el tráfico separadamente para cada una de ellas. Los objetos *Anaports* están reunidos en una lista que

Convstat mantiene como atributo. En la práctica ocurre a menudo que el VFL solo tenga información de una Comunidad y que la lista consista realmente en un solo nodo.

Tras acumular el tráfico de la línea, esta se escribe en el VFL de salida inalterada, salvo que se le añade un token GROUP=*grupo*, donde *grupo* es la clasificación de grupo que se ha hecho a partir de los valores de puerto (si no se pudo clasificar aparecerá 'GROUP=NULL').

El motivo de realizar esta clasificación línea a línea es que puede resultar útil para generar ciertas estadísticas, que cruzan distintos campos del VFL con la clasificación de grupos de aplicación. Por ejemplo, una gráfica que muestre las direcciones IP de los servidores que registran mayor tráfico de determinado grupo de aplicación.

- Clasificación. Una vez reunida la información de tráfico por puertos del fichero VFL en la lista de *Anaports*, hemos de clasificar la información. Esto se realiza invocando *process_TCP* y *process_UDP* en cada uno de ellos, que asignan el tráfico a aplicaciones y grupos de aplicación a partir del tráfico asignado a puertos[3]. Una vez clasificado el tráfico, generamos los ficheros de estadísticas y procedemos con el siguiente *Anaports* (reiniciando previamente a 0 las acumulaciones de tráfico, con *clear*).

  Respecto a la ubicación de los ficheros de estadísticas convencionales, se estructuran en un árbol de directorios, cuya composición sigue la forma: `stat/CA/M**D**H**/`, siendo CA el acrónimo de la Comunidad Autónoma y M**D**H** es una indicación del mes, día y hora correspondiente al fichero VFL analizado y se obtienen a partir de la marca de tiempo en el nombre de este último. Por ejemplo: `stat/VAL/M11D24H03/`.

  Dentro de este directorio se generan tres ficheros:

  - `TG`. Generado por *dump_application_groups*[4] de *Classports*.
  - `MX_TCP`. Generado por *dump* de *Anaports*.
  - `MX_UDP`. Generado por *dump* de *Anaports*.

  Las estadísticas por grupos de aplicación se encuentran en `TG`. El formato de este fichero viene exigido por el mantenimiento de la compatibilidad

[3]Los métodos *process_TCP* y *process_UDP* usan como argumentos respectivamente *ports_TCP* y *ports_UDP* del *Anaports*. Sin embargo, puesto que cada uno de estos vectores ocupa 256 Kbytes de memoria, se realiza un paso por referencia, mediante las funciones *get_TCP* y *get_UDP*.

[4]Alternativamente, puede utilizarse *dump_applications* si se desea obtener estadísticas de aplicaciones.

con los procesos del web dinámico. El fichero consta de varias líneas, de las cuales las primeras son comentarios y son significativas solo las tres últimas:

- línea #NAMES, donde se indican los nombres de los grupos de aplicación. Hay una primera columna 'hora', a la que siguen los nombres de los grupos. Las dos últimas columnas corresponden al tráfico TCP y UDP no clasificado (not_classified) y al de otros protocolos no TCP ni UDP (other_protocols)
- línea #TYPES, que indica el tipo de datos de la tercera línea (todos son bytes, menos la primera columna de la hora).
- línea de valores de tráfico, salvo la primera columna que es una indicación de la hora a la que corresponde el fichero (00, 01, 02, ..., 23)

Por ejemplo, el TG en stat/VAL/M11D24H03/ podría ser:

```
# application group traffic data
# generated by econv
#NAMES hora WWW DNS <...> IMESH ROMNET UNKNOWN not_classified other_protocols
#TYPES nombre bytes bytes <...> bytes bytes bytes  bytes bytes
03 7503006 357650 <...> 109170479 9164 0 487781044 380941
```

Las estadísticas de bajo nivel sobre uso de puertos se encuentran en MX_TCP (para puertos TCP) y MX_UDP (para puertos UDP). Estos ficheros están formados por dos columnas: la primera de ellas indica los índices de puerto y la segunda de ellas el tráfico en dicho puerto. Todos aquellos puertos que no aparezcan tienen un tráfico de 0 bytes.

Una alternativa en el formato de este fichero hubiera sido una única columna con la información de los 65536 puertos. Sin embargo, los ficheros habrían sido mucho más voluminosos, así que nos aprovechamos del hecho de que la mayor parte de los puertos no cursan ningún tráfico.

5.4.3 Estrategias de clasificación

En el apartado 5.2 nos hemos referido a la clasificación sin entrar en detalle de como se realiza. Vamos a describirla con algo más de detalle. En cada línea del VFL tenemos cinco campos de interés para realizar la clasificación

- Protocolo.
- Puerto local.

- Puerto remoto.
- Bytes de entrada.
- Bytes de salida.

Si el protocolo no es ni UDP ni TCP, tan solo nos interesan los bytes de entrada y bytes de salida. Se acumulan ambos como tráfico cursado por 'otros protocolos'.

Si el protocolo es UDP o TCP, asignamos los bytes de entrada al puerto local y los bytes de salida al puerto remoto. Esta es la filosofía general que se sigue en la generación de los informes del web dinámico. Vamos a ver como afecta esto desde el punto de vista de clientes y servidores, considerando la asimetría de tráfico (mucho tráfico de servidor a cliente, poco en sentido contrario), que los puertos de los servidores son concretos y que los de los clientes aleatorios.

- Un cliente local accediendo a un servidor remoto. En este caso, el tráfico de entrada, asignado a un puerto aleatorio, será bajo (poco tráfico enviado desde el cliente al servidor) y el tráfico de salida, asignado al puerto del servicio, será alto.
- Un cliente remoto accediendo a un servidor local. En este caso, el tráfico de entrada, asignado a un puerto concreto, será alto y el tráfico de salida, asignado a un puerto aleatorio, será bajo.

Observar que lo que se consigue en ambos casos es que se registre la mayor parte del tráfico de puertos "interesantes" (aquellos asociados a servicios). Si se usara el criterio opuesto (bytes de salida al puerto local y los bytes de entrada al puerto remoto) el principal volumen de tráfico (de los servidores a clientes) se dispersaría entre los puertos aleatorios, y no resultarían muy útiles las estadísticas.

La clasificación que se realiza línea a línea en el VFL (la obtención del token GROUP=*group*, realizada por el método *group_by_port* del *Classports*) es ligeramente diferente. Si el protocolo no es ni TCP ni UDP, *group* es NULL. En otro caso, se intenta la clasificación a través del puerto local. Si esta falla, se intenta a través del puerto remoto, y si esta también falla, el resultado es NULL. No se utilizan para nada los valores de bytes de entrada y salida.

5.5 Resultados

Sobre la sincronización del módulo de estadísticas convencionales, hemos de tener en cuenta que este se ejecuta en segundo plano y que permanece a la espera hasta que aparece un fichero VFL. Normalmente *vflow* genera uno de estos ficheros cada hora, por lo que en un principio se pensó que se podría programar la generación de estadísticas convencionales con *cron*. Sin embargo, esta alternativa se desestimó más adelante, pues se consideró más fiable usar el mecanismo de sincronización productor-consumidor (apartado 2.3.2), ampliamente utilizado en los demás procesos de MIRA.

En las pruebas que se han realizado en las máquinas de análisis, se ha observado que la exploración de los ficheros VFL resulta bastante costosa, debido al gran volumen de estos ficheros (en torno a los 5 Mbytes cada uno), aunque no llega a superar al cuello de botella que es *anaflow*. En todo caso, la pérdida de eficiencia es el precio que hay que pagar por realizar un diseño no intrusivo del módulo de estadísticas convencionales. Una alternativa, más eficiente pero con mayor coste en modificaciones, es realizar modificaciones en *vflow* para que, según realiza la clasificación por veredicto, generara las estadísticas convencionales (es decir, agrupar la funcionalidad de *vflow* y *econv* en un único proceso).

Los ficheros TG, MX_UDP, MX_TCP resultan bastante ligeros (estando su tamaño medio en torno a 1 Kbyte, 15 Kbytes y 80 Kbytes respectivamente) con lo que el uso de espacio de almacenamiento en las máquinas no se ve comprometido. Además, se trata de ficheros temporales, que son acumulados y consolidados por los procesos del web dinámico para obtener ficheros más definitivos y sobre los que se pueden definir políticas de borrado.

5.6 Conclusiones

El módulo de estadísticas convencionales se diseñó como complemento a las estadísticas de tráfico por análisis de contenidos. Por tanto, se trata de enriquecer los informes ofrecidos al consultor, para que tenga varias "vistas" del tráfico que está cursando la red monitorizada.

En términos de tiempo y esfuerzo de desarrollo así como de carga de proceso que se añaden a la plataforma compensan completamente la funcionalidad obtenida. La alternativa, de no haberse incorporado el módulo de estadísticas convencionales, sería la utilización de un analizador de protocolos en paralelo, con una posible pérdida de eficiencia y perdiendo la ventaja de la integración.

Es de destacar que mediante la información suministrada por CAIDA se han podido desarrollar estadísticas de alto nivel. Sin recurrir a esta información externa, hubiera sido necesario una investigación de las numerosas aplicaciones existentes en Internet actualmente, tarea nada sencilla.

En lo que respecta al uso que se ha dado a las nuevas funciones incorporadas, los informes de medidas presentados ha RedIris desde este desarrollo han incluido análisis de tráfico TCP y UDP (estadísticas de bajo nivel) y de aplicaciones (estadísticas de alto nivel). En estos análisis se han obtenido conclusiones útiles sobre aspectos de la red monitorizada que, de no haber utilizado el módulo de estadísticas convencionales, habrían pasado inadvertidas.

Capítulo 6

Desarrollos adicionales

6.1 Introducción

En los capítulos anteriores se han descrito las principales tareas realizadas para la ampliación de las funcionalidades de la plataforma. No obstante, se desarrollaron también una serie de mejoras que si bien son de menor envergadura, constituyen una parte importante del trabajo realizado.

En concreto, se describirán las mejoras en el proceso de búsqueda de patrones (apartado 6.2), la incorporación de funciones para la medida de utilización de patrones (apartado 6.3) y la descripción de una gramática para la configuración del algoritmo de determinación de veredicto, permitiendo una flexibilización del mismo (apartado 6.4).

6.2 Mejora del algoritmo de búsqueda de patrones

El proceso *anaflow*, encargado de realizar el preprocesado del tráfico capturado, presenta algunas deficiencias en lo que respecta a la búsqueda de patrones para la detección de síntomas.

- El proceso de búsqueda está demasiado orientado a *strings* del estilo utilizado en el lenguaje C. Es decir, cadenas de bytes cuyo final está señalado por un byte con valor 0. Esto presenta un problema: no es posible analizar paquetes que contengan bytes con valor 0, pues esto provocaría un final prematuro en el análisis del contenido del paquete (utilizando funciones como *strstr*, mencionada más adelante).

  Para evitar que ocurra esto, antes de procesar el paquete se recorre cambiando todos los bytes 0 por 32 (valor del espacio en blanco en ASCII).

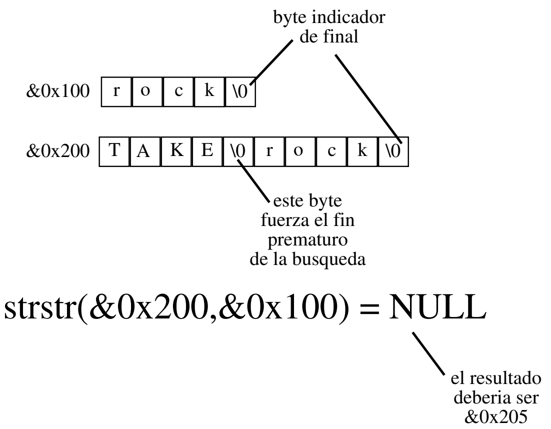


Figura 6.1: Problemas de *strstr* con el byte 0.

Si los patrones a buscar no contienen el byte 0 (como son los textos) esta solución puede utilizarse[1].

El problema se da cuando los patrones a buscar son binarios, algo que inicialmente no se consideró en la plataforma, pero que la detección de MP3 (capítulo 3) y la búsqueda de firmas de ataques de seguridad (capítulo 4) necesitan.

- La búsqueda de los patrones iSTR (aquellos en los que no se distingue entre mayúsculas y minúsculas) es muy ineficiente. La razón de esta ineficiencia es que la función que implementa esta búsqueda utiliza un simple algoritmo de fuerza bruta, computacionalmente costoso[2].

  Un patrón iSTR resulta (en media) unas 35 veces más costoso de analizar que un patrón STR de la misma longitud. Teniendo en cuenta que más de tres cuartas parte de los patrones del fichero de configuración habitual en rule.cfg son de tipo iSTR, esto supone un gran problema de eficiencia.

  En cuanto a STR, se basa en la utilización de la función *strstr* de la librería glibc. A pesar de tratarse de software de libre distribución cuyo código fuente es accesible, el autor no aclara el algoritmo que está utilizando, como puede verse en el siguiente fragmento, que encabeza strstr.c:

[1] Si el patrón contiene blancos, es posible que el cambiar un byte 0 por un 32 se produzca la aparición de un patrón cuando realmente no existía en el paquete original. De todas formas, las posibilidades de que esto ocurra, son muy remotas.

[2] Orden $0(nm)$, siendo n y m las longitudes del contenido donde se busca y del patrón buscado respectivamente.

> My personal strstr() implementation that beats most other algorithms. Until someone tells me otherwise, I assume that this is the fastest implementation of strstr() in C. I deliberately chose not to comment it. You should have at least as much fun trying to understand it, as I had to write it :-).
>
> Stephen R. van den Berg, berg@pool.informatik.rwth-aachen.de

Por tanto, tampoco tenemos garantías de que *strstr* utilice el algoritmo más eficiente para realizar la búsqueda de patrones.

A continuación se describen las modificaciones realizadas sobre la plataforma en ambos aspectos para mejorar globalmente el motor de búsqueda.

6.2.1 Implementación de patrones binarios

En primer lugar, se ha modificado el parser de *anaflow* que lee la configuración, para que interprete un nuevo tipo de patrones: HSTR. Este tipo se utilizará para patrones binarios, indicando el contenido del mismo en el fichero de configuración mediante dígitos hexadecimales. En la tabla 3.8 pueden verse ejemplos.

Se utiliza una nueva forma de almacenar los patrones en memoria. Anteriormente, bastaba con utilizar un *string*, pues el byte final 0 indicaba la longitud. Ahora junto al *string* se usa también un campo adicional (un entero) para indicar la longitud del patrón.

Por último, se elimina el preprocesado de los paquetes que alteraba su contenido cambiando bytes 0 por blancos. Como se señaló, esto plantea problemas al utilizar funciones basadas en *strings* (al estilo C) como *strstr*, cuya solución se dará en el siguiente apartado.

6.2.2 Mejoras en el algoritmo de búsqueda

Buscando principalmente la mejora de los costes de búsqueda de patrones iSTR, se decidió adoptar el algoritmo de Boyer-Moore, conocido por ser uno de los mejores para la búsqueda de patrones y que se describen con detalle en el apéndice D.

Debido a que es un algoritmo bien conocido y utilizado, existen abundantes programas de código abierto que lo utilizan y de los que se pueden obtener implementaciones, por lo que no es necesario la programación directa. En nuestro caso, hemos optado por utilizar las funciones *mSearch* (búsqueda de patrones considerando distinción mayúsculas y minúsculas) y *mSearchCI* (búsqueda de patrones sin considerar distinción) presentes en el código fuente de Snort.

Junto con el patrón de búsqueda, es necesario construir y almacenar la función de desplazamiento que necesita el algoritmo para funcionar. Esta tabla supone un ocupamiento de la memoria que habría que considerar en el caso de utilizar configuraciones con gran número de patrones, pero en general no supone problema.

A diferencia de *strstr*, las funciones *mSearch* y *mSearchCI* no tienen el problema de tener que trabajar obligatoriamente con *strings* tipo C, ya que entre sus argumentos están n y m, las longitudes del contenido y del patrón a buscar en él, respectivamente.

Con el uso de la función *mSearchCI* se ha conseguido una significativa mejora en la eficiencia en torno al 40% en la evaluación de patrones iSTR en el caso peor. La mejora puede ser mayor si se utilizan patrones largos, por las propiedades del algoritmo de Boyer-Moore. En el caso de *mSearch* para la evaluación de patrones STR, la eficiencia es del mismo orden que la que se obtiene con *strstr*[3].

6.3 Medidas del uso de reglas

El conjunto de síntomas que *anaflow* busca en el tráfico capturado, así como los patrones de los que consta cada uno de estos síntomas forma parte de la configuración de la plataforma (fichero rule.cfg). Sin embargo, el administrador no cuenta con ningún mecanismo directo de realimentación que le permita saber la utilidad de estos patrones.

Teniendo en cuenta que el proceso que condiciona la eficiencia de la herramienta es precisamente *anaflow* y que la evaluación de cada patrón implica un determinado coste en tiempo de proceso y carga de trabajo de la CPU, conviene configurar un conjunto de patrones que resulte lo más útil posible. Los patrones son tanto más útiles cuando más ocurrencias (estadísticamente hablando) se encuentran de los mismos dentro del tráfico capturado.

Un patrón inútil (aquel que nunca aparece en el tráfico capturado) es pernicioso, por dos razones:

- Al no encontrarse nunca, nunca producirá un síntoma dentro del flujo y, debido a ello, resulta inútil para realizar la clasificación.

- La búsqueda de un patrón (y del síntoma a él asociado) en un paquete finaliza en cuanto este aparece en el contenido. Si el patrón no va a aparecer nunca, es necesario recorrer el contenido entero, con lo que

[3]Esto demuestra que el autor tenía razón al afirmar que su algoritmo era uno de los más eficientes.

estamos perdiendo el tiempo. El coste (en tiempo de proceso y capacidad de la CPU) es siempre máximo.

Para mejorar esta situación se realizaron modificaciones en *anaflow* con el objetivo de obtener informes sobre el uso de patrones y síntomas. Con esta información, el administrador puede hacerse una idea de la utilidad de los distintos patrones y depurar la configuración.

6.3.1 Modificaciones en *anaflow*

Cuando *anaflow* arranca construye una base de datos interna con el conjunto de síntomas y los patrones de los que consta cada uno. Cada vez que se inspecciona un paquete se hace uso de esta base de datos para saber que patrones buscar.

Las modificaciones que fue necesario realizar fueron las siguientes:

- Incorporación de contadores en los patrones y síntomas. De esta forma, cada vez que un patrón es encontrado en un paquete, su contador se incrementa en 1, al igual que el del síntoma que contiene ese patrón. De esta forma, se tiene una medida de la utilidad: cuanto más alto sea el valor de este contador, mayor será el número de veces que ese patrón ha sido útil para encontrar un síntoma.

  En realidad, estos contadores ya se encontraban en el código de *anaflow*, pero hasta ahora no se había hecho uso de ellos.

- Generación de informes de uso. Para obtener los informes de uso, se decidió utilizar el paradigma de señales a procesos UNIX. De esta forma, cuando el consultor quiere obtener un informe del uso de las reglas en *anaflow* le envía una señal software.

  La respuesta del proceso a esta señal es la generación de un fichero de texto donde aparecen los distintos patrones (agrupados por síntomas) y el valor de sus contadores (denominaremos a este valor *índice de ocurrencias* del patrón). Estos ficheros tienen nombre RUSE0000, RUSE0001, ... y se generan en el directorio especificado por el administrador al arrancar *anaflow*.

  Adicionalmente, *anaflow* genera un fichero RUSEFINAL si por alguna causa debe finalizar su ejecución prematuramente.

Estos ficheros le sirven al administrador para determinar los patrones más utilizados. Teniendo en cuenta la aleatoriedad con la que los patrones se presentan en el tráfico capturado, cuanto más tiempo haya estado funcionando *anaflow* más fiables es la información obtenida.

6.3.2 Depuración de la configuración

Obviando el posible tratamiento estadístico (ignorar informes correspondientes a ejecuciones de *anaflow* muy cortas, sumar el resultado de informes de distintas máquinas, ...) y suponiendo que se parte de información lo más fiable posible, la forma de realizar la depuración de la configuración es bastante evidente. Para cada síntoma:

1. Eliminar aquellos patrones con un índice de ocurrencias igual o por debajo de cierto umbral. Este umbral puede ser 0 (se eliminan solo aquellos patrones completamente inútiles) u otro valor (preferiblemente establecido en función del máximo índice de ocurrencias del conjunto de patrones).

2. Los patrones tipo $\wedge$ se sitúan siempre al principio del fichero de configuración (su coste de búsqueda es prácticamente nulo, puesto que no hay que recorrer el contenido del paquete). El resto, ordenarlos en orden decreciente de índice de ocurrencias. El orden en el que se evalúan los patrones de un síntoma es el orden en el que aparecen en el fichero de configuración, por lo que nos interesa evaluar primero los patrones más útiles (que tendrán una probabilidad mayor de aparecer y por tanto finalizar antes el proceso de búsqueda del síntoma).

En el apartado 3.5.4 se explica como se utilizan trazas de uso RUSE para la depuración de los patrones utilizados en la detección MP3. A continuación, vamos a mostrar un ejemplo adicional. Consideremos que tenemos el siguiente fichero (o agregado de ficheros) RUSE:

```
r1:HTTP (3170)
  iSTR:'HTTP/1.0' (1099)
  iSTR:'HTTP/1.1' (1437)
  iSTR:'HTML>' (30)
  iSTR:'HEAD>' (5)
  iSTR:'TITLE>' (0)
  iSTR:'BODY>' (12)
  iSTR:'<B>' (284)
  iSTR:'</B>' (64)
  iSTR:'<I>' (23)
  iSTR:'</I>' (12)
  iSTR:'<P>' (38)
  iSTR:'</P>' (166)
  iSTR:'<HR>' (0)
```

Supongamos que establecemos como umbral de utilidad 100 ocurrencias. Tras eliminar aquellos patrones considerados inútiles y reordenar el resto, el resultado sería la siguiente configuración en rule.cfg.

HTTP;iSTR;HTTP/1.1	(1437)
HTTP;iSTR;HTTP/1.0	(1099)
HTTP;iSTR;<B>	(284)
HTTP;iSTR;</P>	(166)

6.4 Flexibilización de la clasificación de flujos

El algoritmo para la determinación de veredicto que implementa *vflow* no forma parte de la configuración de la plataforma. Es decir, esta predeterminado y el administrador de la plataforma no puede cambiarlo. Teniendo en cuenta el objetivo de flexibilizar al máximo el uso de la plataforma que se persigue en MIRA, sería deseable que fuera configurable. En concreto, se persiguen los siguientes objetivos:

- Permitir un número arbitrario de categorías de tráfico (en la actualidad son solo cuatro: ACA, LUD, COM e IND) con la semántica que decida el usuario de la plataforma. Utilizar tan solo cuatro categorías puede ser demasiado rígido para algunas situaciones. Por ejemplo, si queremos distinguir entre varios tipos de tráfico lúdico (categoría para juegos, categoría para música, ...).

- Permitir una mayor flexibilidad en el algoritmo de clasificación de flujos (actualmente, fijo y "cableado" dentro del código). Por ejemplo, el algoritmo actual aplica "presunción de inocencia", clasificando como ACA todo el tráfico que no tenga indicios claros de pertenecer a otra categoría. Pudiendo modificar el algoritmo de decisión, podríamos aplicar otro criterio (por ejemplo, considerar todo el tráfico sin indicios como IND).

A continuación se describirá el diseño de una gramática que permite la especificación de un algoritmo más flexible, cumpliendo estos dos objetivos. La implementación definitiva, no realizada, se contempla como una posible línea de trabajo futuro en el apartado 7.2.7.

6.4.1 Definición de la gramática

Es el administrador el que describe el algoritmo de clasificación que quiere usar en un fichero de configuración (equivalente al `verdict.cfg` actual). La especificación formal de este fichero de configuración consiste en la gramática de la figura 6.2.

<configuración> ⟶ <lista_CCAA><lista_conjuntos><lista_reglas>
|<lista_CCAA><lista_reglas>

<lista_CCAA> ⟶ <CCAA>
|<lista_CCAA><CCAA>

<CCAA> ⟶ *num/num*

<lista_conjuntos> ⟶ <conjunto>
|<lista_conjuntos><conjunto>

<conjunto> ⟶ <token> = <lista_tokens>

<lista_tokens> ⟶ (<secuencia_tokens>)

<secuencia_tokens> ⟶ *token*
|<secuencia_tokens>,*token*

<lista_reglas> ⟶ <regla>
|<lista_reglas><regla>

<regla> ⟶ <condiciones> `->` *token*

<condiciones> ⟶ <condiciones> AND <condiciones>
|<condiciones> OR <condiciones>
|NOT <condiciones>
|(<condiciones>)
|<condición>

<condición> ⟶ PORT <rango_puertos>
|MAJORITY *token*
|MAJORITY
|DEFAULT
|<literal> = <literal>
|<literal> > <literal>
|<literal> < <literal>
|<literal> `>=` <literal>
|<literal> `<=` <literal>

<literal> ⟶ *token* | *num*

Figura 6.2: Gramática para la flexibilización de *vflow*.

El lexema *token* es una cadena alfanumérica de longitud arbitraria donde el primer carácter debe ser una letra[4]. El lexema *num* es un número entero positivo, incluyendo el cero[5].

Como puede verse, el fichero de configuración consta de tres secciones diferenciadas:

- Una lista con la definición de los VPI/VCI correspondientes a cada Comunidad Autónoma, usada para la generación de informes finales, pero que no interviene en el proceso de determinación de veredicto.

- Una forma de definir grupos de síntomas. Esto sirve para agregar varios síntomas conjuntamente a la hora de evaluarlos en una condición aritmética. Es opcional.

- Una lista de reglas. La parte más importante. Comenzando por la primera regla definida, se comprueba secuencialmente si el flujo cumple alguna de ellas, en cuyo caso ya lo tenemos clasificado y pasamos al siguiente flujo.

  Una regla está formada por una agrupación de condiciones. Existen condiciones atómicas que pueden ser unidas con conectores de tipo lógico (además, observar que la secuencia de reglas dentro del fichero constituye un OR implícito).

  En cuanto a las condiciones atómicas, tenemos las siguientes:

  - PORT. Verdadera si alguno de los puertos del flujo pertenece al rango especificado. El rango puede constar de un único puerto.
  - MAJORITY *token*. Donde *token* es un síntoma. Verdadera si el síntoma *token* es mayor que cualquier otro.
  - MAJORITY. Verdadera si existe un síntoma que sea mayor que cualquier otro (si no es verdadera, hay un empate al menos entre los dos síntomas más mayoritarios).
  - DEFAULT. Siempre verdadera. Sirve para definir la última regla del fichero, clasificación por defecto.
  - *literal1* <op_aritmético> *literal2*. Comprueban que la relación aritmética es verdadera. Si *literal1* o *literal2* son conjuntos de síntomas,

[4] La única restricción es que no coincida con alguna de las palabras reservadas, como AND o NOT.

[5] Las líneas del fichero de configuración que comienzan por # se consideran comentarios y son ignoradas. No forman parte de la definición de la gramática.

considera el agregado de todos los síntomas que lo componen. Observar que es posible utilizar síntomas, conjuntos de síntomas o constantes numéricas.

La configuración que se muestra en la figura 6.3 reproduce el algoritmo original de funcionamiento[6] (figura 2.10). Esto demuestra que la gramática definida es compatible con hacia atrás.

```
# Lista de CCAA
111/111 CAL
111/112 CAT
...

# Lista de conjuntos de síntomas
sintomas_LUD = {IRC,SEX,DROGAS,JUEGOS,SPORT,MUSICA}
sintomas_COM = {SHHTP,HTTPS,APUESTA,...,AUTHENTICATE,BANCA}
sintomas_ACA = {HTTP,FTP,NEWS,MAIL,TELNET,RLOGIN}

# Lista de reglas

(sintomas_COM > 0) AND sintomas_COM >= sintomas_LUD -> COM
sintomas_LUD > sintomas_COM -> LUD

majority HTTP AND port 80 -> ACA
majority FTP AND port 20-21 -> ACA
majority NEWS AND port 119 -> ACA
majority MAIL AND port 25 -> ACA
majority TELNET AND port 23 -> ACA
majority RLOGIN AND port 513 -> ACA

majority -> IND

port 6667 -> LUD

default -> ACA
```

Figura 6.3: Ejemplo de uso de la gramática.

[6]En lo que respecta a *vflow*. La clasificación en función de las direcciones que realiza *anaflow* queda excluida.

Capítulo 7

Conclusiones y futuros trabajos

7.1 Conclusiones

El trabajo que se describe en este proyecto, ha contribuido de forma significativa al progreso del proyecto MIRA, en concreto a las actividades de detección avanzada, estadísticas convencionales e incremento de las tasas de captura dentro del paquete de consolidación de MEHARI y el paquete de ataques de seguridad por entero.

A diferencia de otro tipo de proyectos, donde se realiza un desarrollo partiendo de cero, en nuestro caso el objetivo era realizar un conjunto de mejoras sobre una plataforma ya existente. Por tanto, la primera actividad que hubo que afrontar fue el estudio pormenorizado de la plataforma en sí, cuyo fruto dentro de este documento lo constituye el capítulo 2.

En relación a ello, la principal dificultad fue el estudio y compresión del código fuente de los distintos programas, dada su extensión (existen multitud de procesos) y la ausencia generalizada de comentarios en el mismo. Sin embargo, una vez realizada esta tarea, las modificaciones en la plataforma se simplifican, ya que sabemos en todo momento que es lo que hay que modificar para conseguir los objetivos deseados. Adicionalmente, permite un alto grado de reutilización, mediante el uso de librerías comunes que cumplen funciones útiles en todos los procesos (por ejemplo, el paradigma de comunicación productor-consumidor de ficheros está implementado en una de esta librerías, por lo que, a la hora de programar *snortflow* lo único que hubo que hacer fue incluirla).

Otra de las diferencias significativas con respecto a otro tipo de proyectos más "clásicos" donde el desarrollo sigue unas fases claras de análisis, diseño, implementación y pruebas, es que en nuestro caso tenemos distintas actividades con relativa independencia entre ellas. De hecho, el desarrollo de las mismas ha sido concurrente en el tiempo a lo largo de la realización del proyecto, principalmente por necesidades de sincronización con el resto del equipo de trabajo del proyecto MIRA.

Por tanto, la mejor manera de entender el trabajo realizado es como un conjunto de subproyectos, cada uno con sus propias fases de análisis, diseño, implementación y pruebas.

- En el capítulo 3 se describe el desarrollo de técnicas avanzadas para la detección de tráfico lúdico. Esta actividad no estaba prevista inicialmente como parte del MIRA, pero se realizó su inclusión debido al fuerte crecimiento de este tipo de tráfico en Internet y la necesidad asociada de conocer su dimensión de forma precisa. En concreto, se consideró el audio MP3, aunque la metodología empleada puede utilizarse para la caracterización de otros tipos de tráfico lúdico.

  En primer lugar, se realizó un análisis del formato MP3, ya que, aunque su utilización está muy popularizada, es necesario una tarea de investigación para entender como está codificado el audio. También se analizaron las aplicaciones P2P con las que este audio es distribuido. Posteriormente, se pasó a la fase de diseño, en la que se establecieron heurísticos basados en el contenido de los paquetes, siguiendo la filosofía de inspección de paquetes del MIRA. El siguiente paso fue la implementación de estos mecanismos en la plataforma y, finalmente, tuvo lugar la recogida de resultados y la depuración del conjunto de los heurísticos.

- En el capítulo 4 se aborda el desarrollo de las funciones de seguridad. En primer lugar, se realizó un estudio de como encaja una plataforma de las características de MIRA en el marco de la gestión de seguridad de red, ya que, a diferencia de otros elementos más típicos (como un cortafuegos) no estaba del todo claro en un principio. Como resultado, se establecieron los objetivos técnicos a cumplir.

  Dentro de las tareas de análisis, fue muy importante el estudio de Snort y sus herramientas asociadas, un sistema de detección de intrusiones para redes locales con unas características muy similares a las que se querían incorporar en MIRA.

  Posteriormente, tuvo lugar la fase de diseño, con la evaluación de las distintas alternativas, adoptándose finalmente aquella que más posibilidades presentaba, aunque más compleja en su realización. En todo caso, se buscó minimizar el impacto en la plataforma ya existente. La implementación de la nueva arquitectura incluyó el desarrollo de nuevos procesos (*cap2pcap* y *snortflow*), así como el desarrollo de nuevos mecanismos de consolidación de la información.

Aunque la filosofía del MIRA se basa en la monitorización pasiva y las principales funcionalidades de seguridad se encuentran en esta línea, también se ha incorporado un mecanismo activo capaz de efectuar acciones de respuesta ante ciertos ataques, considerados críticos. Estas funciones pueden resultar útiles en un escenario de red local.

- El capítulo 5 describe el módulo de estadísticas convencionales de la plataforma. Tras un análisis de lo que se quería conseguir, distinguiendo entre dos tipos de estadísticas, a bajo y alto nivel, el trabajo ha consistido en el diseño, implementación y pruebas de un proceso para la generación de estas estadísticas, *econv*.

  Este proceso incorpora una estructura de clasificación de tráfico flexible, que permite extraer información a distintos niveles. Dentro de su desarrollo, se ha cuidado su integración dentro de la plataforma, respetando las interfaces de entrada y salida de los distintos procesos con los que se comunica. La utilidad de este módulo ha quedado probada en los informes de medidas presentados a RedIris, donde parte de las estadísticas han sido generadas con *econv*.

- Adicionalmente a estas tres grandes tareas, se realizaron un conjunto de modificaciones menores, destinadas a mejorar el motor de análisis (tanto por la inclusión de nuevos tipos de patrones de búsqueda como por la mejora de su eficiencia) como en el sistema de clasificación de tráfico, que se describen en el capítulo 6.

Queremos destacar el uso e incorporación que se ha hecho de resultados de proyectos externos a MIRA, evitando la ineficiencia que supondría realizar el mismo trabajo desde cero. En concreto, la inclusión de Snort como motor de análisis con funciones de seguridad incorporadas y de su librería de búsqueda de patrones y la utilización de la configuración de la herramienta [CoralReef] de CAIDA.

Desde el punto de vista de la ingeniería del software, esta clase de *outsourcing* consistente en la reutilización de desarrollos de libre distribución, resulta muy conveniente, tanto desde el punto de vista del proyecto MIRA (pues simplifica enormemente la resolución de un problema complejo) como desde el punto de vista del desarrollo externo utilizado (ya que demuestra nuevas posibilidades de utilización del mismo y, por tanto, enriquece el abanico de aplicaciones de la herramienta).

Por último, señalamos las principales ventajas que aporta la versión definitiva de MIRA, con respeto a otras alternativas de plataformas de captura y monitorización.

- Inspección de contenidos. La monitorización de tráfico se realiza mediante la inspección del contenido de los paquetes. Esto supone una ventaja frente a las alternativas que se basan tan solo en la inspección de las cabeceras de los paquetes, ya que estas últimas no están analizando realmente el tráfico cursado en la red. Por ejemplo, si un servidor web utiliza el puerto 21712 en vez del 80, una de estas plataformas no podrá detectar ese tráfico como HTTP, mientras que MIRA sí.

- Bajo coste. Al estar basada la plataforma en equipos PC convencionales y sistemas operativos libres, en los que no es necesario pagar por una licencia de uso, su coste es bajo.

- Modularidad. La plataforma está compuesta por un conjunto de módulos o procesos que realizan funciones independientes entre sí. De esta forma, pueden desarrollarse nuevas funcionalidades sin alterar significativamente la arquitectura. Adicionalmente, la modularidad permite una fácil escalabilidad, mediante la replicación de aquellos módulos que sean necesarios.

- Flexibilidad. En parte como consecuencia de lo anterior, la plataforma puede ser adaptada a distintos escenarios de aplicación (apartado 2.4), tan diversos como una red de transporte ATM o una red local Ethernet. Adicionalmente, los distintos procesos admiten posibilidad de ser configurados a medida de las necesidades concretas de utilización.

- Integración. La plataforma proporciona de manera integrada funciones de análisis de paquetes, estadísticas convencionales y seguridad. Esto constituye un verdadero valor añadido a lo que sería utilizar tres plataformas distintas en paralelo (cada una de ellas especializada en una función), principalmente desde el punto de vista de la eficiencia (desperdicio de recursos de análisis no compartidos entre las tres).

7.2 Futuros trabajos

A pesar de que el resultado del trabajo realizado es satisfactorio, todo proyecto de ingeniería puede ser mejorado y extendido. A continuación se describen posibles líneas de actividad que podrían aumentar las posibilidades de la plataforma MIRA.

7.2.1 Correlación avanzada de flujos

El proceso *biflow* de la versión actual de la plataforma realiza la correlación de flujos unidireccionales de entrada y salida para la obtención de biflujos (apartado 2.3.1.4). Sin embargo, esto es una limitación impuesta principalmente por el escenario de aplicación ATM de RedIris, donde existen dos fibras con sentidos de tráfico diferenciados.

En un escenario de red local Ethernet, la necesidad de correlación bidireccional obliga a definir los flujos en una semántica equivalente a "entrada y salida". El flujo de salida es todo aquel con origen fuera de la red y destino dentro de ella, y el flujo de salida viceversa. Sin embargo, existen otro tipo de flujos de tráfico que se producen (tráfico interno, con origen y destino en la propia red y tráfico en tránsito, originado y destinado fuera de la red) a los que forzosamente hay que asociar con un sentido de salida o entrada si queremos que sean capturados y analizados. El mismo problema surge en escenarios complejos de arquitectura distribuida (como el planteado como ejemplo en el apartado 7.2.2).

Para solucionar el problema, la correlación debería de realizarse sobre todo el conjunto de flujos unidireccionales. Es decir, se examina todo el conjunto del tráfico (sin distinguir entre una categoría de entrada y otra de salida) buscando flujos tales que la dirección y puerto origen de uno sea la dirección y puerto destino del otro y viceversa.

Una posibilidad más interesante aún sería la correlación de varios flujos bidireccionales entre sí, cuando se encuentren relacionados. Por ejemplo, si se detecta que un flujo de audio MP3 dirigido desde el servidor A usando el puerto $p1$ hacia el cliente B puerto $p2$, podría correlarse como tráfico lúdico todos aquellos flujos cuyo origen fuera A $p1$, independientemente del destino, ya que por lo que parece estamos ante un servidor de *streaming*. No obstante, descubrir este tipo de relaciones sin equivocarnos requiere capacidades complejas en el proceso *biflow*, por lo que podría resultar difícil llevarlo a cabo.

7.2.2 Arquitectura distribuida

Como se ha señalado en las conclusiones, la arquitectura modular presente en MIRA permite una fácil escalabilidad. Esta escalabilidad puede utilizarse de dos maneras. Por una parte, para aumentar el número de sensores de tráfico situados en un mismo punto. Esto permite resolver problemas de eficiencia que se tendrían con un solo sensor, permitiendo de esta forma la captura y análisis de un mayor volumen de tráfico.

Otra forma de aprovechar esta escalabilidad es realizar un despliegue distribuido de la plataforma, situando sensores en distintos puntos de la red mo-

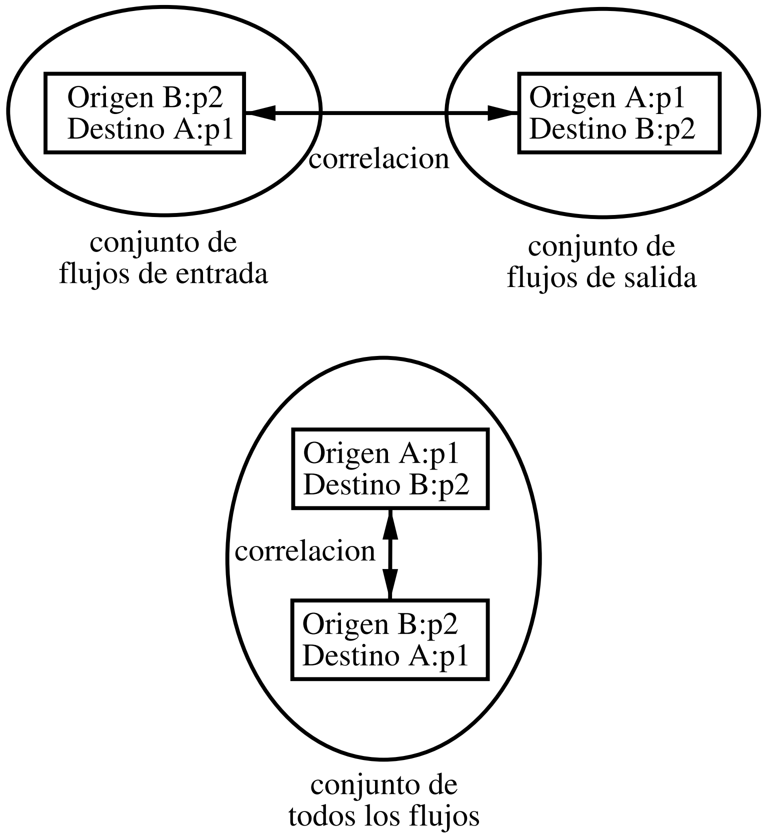


Figura 7.1: Alternativas de correlación de flujos actual (arriba) y óptima (abajo).

nitorizada. Consideremos como ejemplo la red de la figura 7.2.

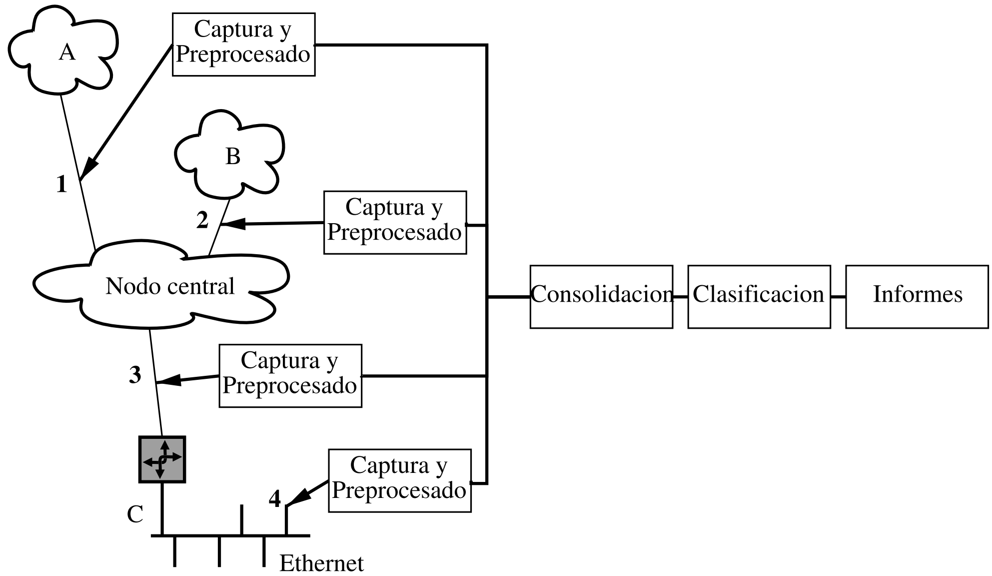


Figura 7.2: Ejemplo de uso distribuido de la plataforma MIRA.

Analizamos los distintos problemas que pueden surgir y cual es la solución a cada uno.

- Hay que evitar que un mismo paquete sea capturado varias veces por distintos sensores. Por ejemplo, un paquete en la red A con destino a la red B, sería capturado por los sensores 1 y 2. Esto se soluciona con una adecuada configuración en cada sonda sobre que tráfico debe capturar.

  En este ejemplo, las sondas 1, 2 y 3 se configuran para que capturen solo aquel tráfico proveniente del nodo central (independientemente de su origen) y con destino a la red A, B o C (respectivamente) y el tráfico con origen en la A, B o C (respectivamente) y destino el nodo central. Con respecto a la sonda 4, se configura para que solo capture el tráfico interno a la red C (el resto del tráfico lo capturará una de las otras sondas).

- La correlación no puede realizarse localmente, puesto que existen distintas direcciones del mismo tráfico en distintos enlaces. La correlación debería realizarse después de la consolidación. Por tanto, habría que introducir cambios en los procesos que actualmente realizan estas funciones (*biflow* y *mflow*).

- En la implantación en RedIris los procesos en distintas máquinas se comunican por NFS mediante una Ethernet dedicada. Cuando las máquinas se encuentran separadas entre sí una gran distancia, de forma que no puede utilizarse una red local, pueden existir retardos y pérdidas de paquetes[1] que introduzcan un cuello de botella en el procesado.

Observar como es posible tener una red compleja que utilice distintas tecnologías sin que esto afecte al diseño distribuido, supuesto que existen capturadores para los distintos casos. Actualmente, en MIRA existen capturadores tanto para ATM como para Ethernet. Nuevas posibilidades a ser desarrolladas podrían ser POS (*Packet Over Sonet*) y Gigabit Ethernet.

7.2.3 Aplicación de técnicas de *data mining*

El objetivo final de la herramienta MIRA es la generación de informes para el consultor. No obstante, podemos considerar su utilización en técnicas de *data mining*. Esta rama de la Inteligencia Artificial se especializa en la obtención de patrones de comportamiento sobre un gran volumen de información. Los objetivos pueden ser los siguientes:

- Realimentar la configuración de los procesos de la plataforma (figura 7.3). Se trata de conseguir que mediante la inspección de la información generada por la plataforma se extraigan nuevos patrones y direcciones que permitan afinar o hacer más eficiente la clasificación. Por ejemplo, podría descubrirse que cierto host está generando gran cantidad de tráfico lúdico, por lo que se podría incluir su dirección IP dentro de address.cfg directamente. De esta forma, la base de datos de configuración se va modificando automáticamente, liberando de esta tarea al administrador, que actualmente tiene que realizar las modificaciones manualmente.

- Descubrir patrones ocultos dentro del tráfico que puedan escapar al consultor. Mediante la aplicación de técnicas de segmentación (*clustering*) pueden sacarse conclusiones del tipo “de 13 a 15 horas el servidor A registra una gran cantidad de tráfico en el puerto p de tipo COM” que no saltan a la vista por una simple inspección de los informes. De todas formas, habría que estudiar previamente la dispersión de los datos, ya que, si esta es elevada, estas técnicas no funcionan.

[1]NFS usa UDP como protocolo de transporte.

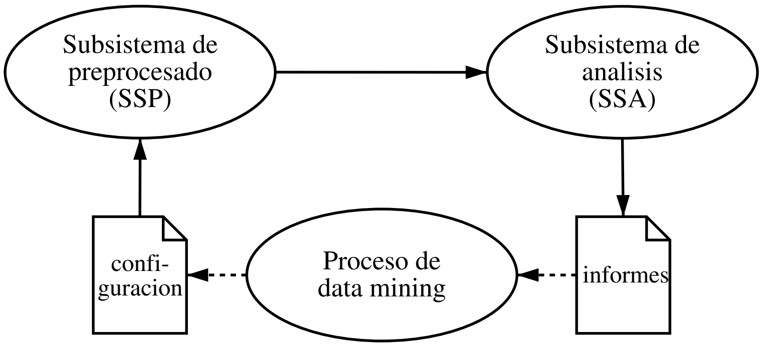


Figura 7.3: Realimentación de la configuración mediante *data mining.*

7.2.4 Interfaz de gestión

En la actualidad, las modificaciones en la plataforma han de ser realizadas manualmente por el administrador, mediante la edición directa de los ficheros donde se encuentra la configuración. Sería deseable facilitar la gestión de la plataforma.

Por ello, una posible mejora sería el desarrollo de una interfaz, preferiblemente gráfica, mediante la cual el administrador pudiera realizar la gestión de la configuración de la plataforma. Podría estar basado en web e integrado con la interfaz de consulta de informes.

7.2.5 Mejoras en las funciones de seguridad

En el apartado 4.6 se mencionan algunos defectos en las funciones de seguridad incorporadas a MIRA, cuya mejora se podría considerar como trabajo futuro.

En primer lugar, sería conveniente medir el impacto (en lo que a eficiencia se refiere) de las nuevas funciones incorporadas y, en función de las medidas, depurar el conjunto de firmas de seguridad que la plataforma detecta.

El proceso *snortflow* también presenta una ineficiencia, al realizar la invocación de Snort mediante llamadas externas a la *shell*, una por cada fichero (figuras 4.13 y 4.15). Existe la posibilidad de realizar un nuevo *snortflow* partiendo del código fuente de Snort (que es de libre distribución) y que solucione los inconvenientes que tiene el uso directo de esta herramienta en la plataforma MIRA:

- Modificar el comportamiento en la comprobación de las reglas. Con las reglas de seguridad el comportamiento sería el de Snort, pero para las reglas de búsqueda de patrones el proceso no debería de parar una vez encontrado uno, sino pasar al siguiente síntoma.

- Modificar la interfaz de entrada, de forma que Snort pueda funcionar en modo *online* consumiendo ficheros PCAP, sin tener que ser reiniciado cada vez que se procesa uno (como es su comportamiento por defecto).

- Modificar la interfaz de salida, de forma que la información sobre los flujos de tráfico convencionales (en los que no se detectaron firmas de ataques) se escriba en un fichero AFL.

Por último, una interesante posibilidad es la actualización automática del conjunto de firmas de seguridad, ya que continuamente aparecen nuevos ataques y vulnerabilidades, y conviene estar lo más al día posible. La base de reglas de Snort es accesible por CVS, por lo que podría programarse la plataforma para que, cada cierto tiempo, accediera al repositorio y descargara las nuevas reglas.

7.2.6 Máscaras binarias en la detección de patrones

Si bien la incorporación de patrones binarios descrita en 6.2 supone ya un gran avance, sería interesante la posibilidad de introducir máscaras binarias para la comparación precisa de los bits dentro de cada byte. Una funcionalidad de este tipo permitiría simplificar la configuración de patrones como los utilizados para la detección de MP3 (tabla 3.5) sin necesidad de tener que introducir todas las combinaciones de bits no significativos de la máscara.

Para ello, incluiríamos un nuevo tipo de patrón, a utilizar en rule.cfg. Por ejemplo, el siguiente patrón se corresponde con el de la primera fila de la tabla 3.5.

```
# <síntoma>;BHSTR;<patrón>;<máscara>
MP3;BHSTR;FFFA90;FFFEFC
```

La implementación puede realizarse de dos formas.

1. No se realiza modificación al algoritmo de búsqueda. El parser de *anaflow* traduce internamente cada patrón BHSTR a un conjunto de patrones HSTR. Se consigue comodidad a la hora de hacer la configuración pero no se mejora la eficiencia en la búsqueda del patrón.

$$\text{MP3;BHSTR;FFFA90;FFFEFC} \longrightarrow \begin{cases} \text{MP3;HSTR;FFFA90;} \\ \text{MP3;HSTR;FFFA91;} \\ \text{MP3;HSTR;FFFA92;} \\ \text{MP3;HSTR;FFFA93;} \\ \text{MP3;HSTR;FFFB90;} \\ \text{MP3;HSTR;FFFB91;} \\ \text{MP3;HSTR;FFFB92;} \\ \text{MP3;HSTR;FFFB93;} \end{cases}$$

2. Modificando el algoritmo de búsqueda. Utilizando el algoritmo de Boyer-Moore (apéndice D), en cada una de las comparaciones habría que realizar una multiplicación lógica (AND) entre la máscara y los datos y comparar el resultado con el patrón. Esta modificación es más compleja que la anterior, pero conseguiría una mayor eficiencia, pues no usamos más patrones que los necesarios.

7.2.7 Implementación de la flexibilización de flujos

En el apartado 6.4 se describe una gramática para la flexibilización del algoritmo de determinación de veredicto, planteándose como trabajo futuro la implementación. Las modificaciones afectan a los siguientes aspectos de la plataforma:

1. *vflow*. Necesidad de incluir un parser que entienda la gramática de la figura 6.2 en el fichero de configuración. En función de este fichero, se creará una lista de árboles de decisión. Cada flujo será presentado secuencialmente a los árboles de la lista, hasta que alguno de ellos dé resultado verdadero y el flujo quede clasificado. La implementación de este parser se puede realizar empleando los analizadores léxico lex y sintáctico yacc.

   El trabajo a realizar en este aspecto es parecido al realizado en [Romeral 01] en el capturador Ethernet, que define una gramática formalmente similar, aunque para un propósito completamente distinto.

   Gran parte del código de evaluación de los nodos de los árboles se encuentra ya presente dentro del propio código de *vflow*, si bien es necesario identificarlo y reutilizarlo convenientemente.

2. Los ficheros VFL no se ven afectados, salvo que ahora el valor de veredicto puede ser cualquier token. Los ficheros REP sin embargo ahora

pueden tener un número arbitrario de categorías de tráfico. Por tanto, su estructura se ve modificada: cada línea deberá añadir tantos campos como sean necesarios según el número de categorías que tengamos. El nombre de las categorías podría indicarse en la primera línea del fichero (actualmente se hace esto, aunque no es necesario, con una línea comentada #).

3. Como consecuencia de la modificación de los REP, será necesario modificar *mflow* y *sumrep* para que puedan entenderlos. En todo caso, estas modificaciones son de menor envergadura que las de *vflow*. Básicamente es pasar de estructuras estáticas (matrices de cuatro elementos, tantos como categorías de tráfico) a estructuras dinámicas (una lista).

Apéndice A

Formato de los ficheros

A.1 Ficheros CAP

Los ficheros CAP tienen formato binario. Al comienzo del fichero hay un byte que indica la versión del formato (actualmente, este byte vale 0). Posteriormente, se suceden una serie de *tramas*, hasta finalizar el fichero. Cada trama consta de los siguientes campos (sucesivos).

Número 4 bytes. Indica el número de trama. La primera trama del fichero tiene número 0, la siguiente número 1 y así sucesivamente. Mediante este campo puede verificarse la integridad del fichero.

Segundos 4 bytes. Instante (en segundos) en el que se capturó la trama, en tiempo UNIX[1].

Microsegundos 4 bytes. Indica los microsegundos de *offset* (positivo) con respecto al instante anterior.

VPI 4 bytes. VPI del circuito donde se capturó la trama.

VCI 4 bytes. VCI del circuito donde se capturó la trama.

Longitud IP 4 bytes. Longitud del datagrama IP capturado.

Longitud AAL5 4 bytes. Longitud de la trama AAL5 capturada. Observar que esta longitud no coincide con la anterior, siendo mayor que ella. Esto es debido a que el nivel AAL5 encapsula el datagrama IP añadiendo un *trailer* de 8 bytes. Si además se usa encapsulado LLC/SNAP, se añaden otros 8 bytes de cabecera (ver detalles en [RFC 1483]). Además, para

[1] Tiempo transcurrido desde las 00:00:00 GMT del 1 de Enero de 1970.

poder ser fragmentado por el nivel SAR en células ATM, se añade relleno para que la longitud sea múltiplo de 48 bytes. Ver apartado 2.4.1 para más detalles sobre AAL5.

Formalmente, es fácil deducir que la relación entre ambas longitudes es:

$$\text{Longitud AAL5} = \text{Longitud IP} + N + P$$

Siendo N igual a 8 en el caso de encapsulado basado en multiplexación de circuito virtual, 16 en caso de LLC/SNAP, y P el valor entero más pequeño (cero incluido) que hace que la longitud AAL5 sea un múltiplo de 48.

Realmente, las aplicaciones que usan los ficheros CAP están tan solo interesadas en el datagrama IP, ignorando los campos adicionales del encapsulado AAL5. Sin embargo, el campo longitud AAL5 es necesario, para delimitar tramas.

Volcado trama tamaño indicado por el campo longitud AAL5. Volcado binario de la trama AAL5 capturada.

Lo que se ha descrito hasta ahora son las tramas ordinarias que contienen paquetes capturados. Existe un tipo distinto, la trama de tiempos, del cual solo existe un ejemplar, que se encuentra al final del fichero, detrás de todas las tramas ordinarias, y que consta de los siguientes campos (sucesivos):

Número 4 bytes. Para señalar el fin de fichero con la presencia de la trama de tiempos, este campo tiene un valor negativo que rompe la secuencia.

Tiempo de inicio (segundos) 4 bytes. Instante (en segundos) en el que se comenzó la captura de tramas para ese fichero, en tiempo UNIX.

Tiempo de inicio (microsegundos) 4 bytes. Indica los microsegundos de *offset* (positivo) con respecto al instante anterior.

Tiempo de finalización (segundos) 4 bytes. Instante (en segundos) en el que se finalizó la captura de tramas para ese fichero, en tiempo UNIX.

Tiempo de finalización (microsegundos) 4 bytes. Indica los microsegundos de *offset* (positivo) con respecto al instante anterior.

Tiempo de captura 8 bytes, interpretados como un *double*. Tiempo efectivo durante el que se estuvo capturando paquetes, expresado en segundos.

A.2 Ficheros PCAP

Los ficheros que usa la librería libpcap (de ahora en adelante, ficheros PCAP) para escribir o leer paquetes capturados son conceptualmente similares a los CAP, puesto que cumplen funciones similares. No obstante, como se verá a continuación el formato es diferente.

Los ficheros PCAP tienen formato binario y constan de una cabecera global, a la que siguen los paquetes, estando compuesto cada uno de ellos por una cabecera de paquete seguida inmediatamente por un volcado de los datos del paquete.

La cabecera global consta de de los siguientes campos.

Número mágico 4 bytes. Consiste en la palabra 0xA1B2C3D4, que identifica el tipo de fichero libpcap y que, adicionalmente, sirve a los programas que lo manejan para saber cual es el convenio de representación de la información binaria (*big-endian* o *little-endian*) en la máquina donde se ejecutan[2].

Número de versión (mayor) 2 bytes. Debe tener valor 2.

Número de versión (menor) 2 bytes. Debe tener valor 4.

Desplazamiento de zona horaria 4 bytes. No usado. Debería valer 0.

Precisión de marca de tiempo 4 bytes. No usado. Debería valer 0.

Intervalo de captura 4 bytes. Longitud máxima de captura en cada paquete, en bytes. Paquetes de mayor longitud, son truncados. Un valor bastante habitual es 65535, con lo que se capturan los paquetes enteros.

Tipo de nivel de enlace 4 bytes. Indica el nivel de enlace sobre el que se capturaron los paquetes. Las posibilidades actuales incluyen:

0 Dispositivos de loopback BSD (anteriores a OpenBSD)

1 Ethernet

6 IEEE 802.5 Token Ring

7 ARCnet

8 SLIP

[2] Existen algunos formatos de libpcap que difieren en algunos campos de los aquí expuestos y que se identifican por un número mágico 0xA1B2CD34. Sin embargo, puesto que no forman parte del estándar de libpcap, no los describiremos aquí.

9 PPP

10 FDDI

100 ATM encapsulado con LLC/SNAP

101 IP directo (*raw*), sin nivel de enlace

102 BSD/OS SLIP

103 BSD/OS PPP

104 Cisco HDLC

105 Red inalámbrica IEEE 802.11

108 Dispositivos de loopback OpenBSD

113 Captura "fabricada" en Linux

114 LocalTalk

Uno de los valores más habituales es 1 (red Ethernet).

La cabecera de paquete consta de los siguientes campos:

Segundos 4 bytes. Instante (en segundos) en el que se capturó el paquete, en tiempo UNIX.

Microsegundos 4 bytes. Indica los microsegundos de *offset* (positivo) con respecto al instante anterior.

Longitud capturada 4 bytes. Indica la longitud capturada del paquete (equivalentemente, la longitud de los datos que existen tras finalizar la cabecera hasta que comienza la siguiente).

Longitud real 4 bytes. Indica la longitud real del paquete. Observar esta es mayor o igual que la anterior y que no coinciden necesariamente (salvo que estemos capturando los paquetes enteros, con un valor adecuadamente grande de intervalo de captura).

A.3 Ficheros AFL

Un fichero AFL representa información de un conjunto de flujos unidireccionales. Está formado por un conjunto de líneas (en ASCII), cada una de las cuales identifica un flujo. En concreto, cada línea (flujo) consta de los siguientes campos, separados por espacios.

VPI/VCI. Circuito ATM en el que se capturaron los paquetes del flujo. Identifica a una Comunidad Autónoma.

Protocolo. Protocolo de nivel de transporte utilizado (según [RFC 1700])

Dirección IP origen.

Puerto origen. Solo tiene sentido cuando el protocolo es TCP o UDP.

Dirección IP destino.

Puerto destino. Solo tiene sentido cuando el protocolo es TCP o UDP.

Bytes cursados . La suma de los bytes de todos los paquetes que componen el flujo.

Número de paquetes. Paquetes cursados por el flujo.

Lista de síntomas. Síntomas detectados en el flujo (separados por';'). Observar que la lista puede estar vacía.

Veredicto. Categoría de tráfico en la que se ha clasificado el flujo (ACA, COM, LUD o IND). Existirá si se ha realizado clasificación anticipada en *anaflow* (basada en address.cfg).

Pueden agregarse líneas de comentario con #, que son ignorada por los procesos que tratan este tipo de ficheros.

Al final del fichero aparecen tres líneas indicando los instantes de inicio y finalización de la captura (tiempo UNIX), así como el intervalo de tiempo durante el cual se realizó captura efectiva de paquetes (segundos).

Ejemplo de un fragmento de fichero AFL[3]:

```
212/169 6 192.168.93.4 65453 192.168.58.67 80 160 4
212/169 6 192.168.120.4 53885 192.168.170.6 25 148668 137  LUD
212/169 6 192.168.120.4 52233 192.168.49.7 25 9384 8 SEX=1;
212/169 6 192.168.120.4 53365 192.168.193.56 25 172 4
212/169 6 192.168.120.4 25 192.168.92.100 1035 80 2
212/169 6 192.168.120.4 51334 192.168.113.142 25 1500 1
212/169 6 192.168.120.4 80 192.168.246.177 3522 6000 4
212/169 6 192.168.2.5 4942 192.168.17.67 80 1071 3 HTTP=3; LUD
212/169 6 192.168.2.5 3919 192.168.97.200 80 2552 8 HTTP=8;SEX=8;
212/169 6 192.168.120.6 443 192.168.98.72 63334 1797 2
212/169 6 192.168.120.6 443 192.168.148.220 1490 14630 14
init_time=1014818400.552627
final_time=1014821997.247040
cap_time=140.142182
```

[3] Tanto en este como en otros ejemplos, se muestran direcciones privadas (identificadas por el prefijo 192.168.0.0/16), que no tienen sentido en una red troncal, por razones de confidencialidad. Igualmente, los valores de VPI/VCI son ficticios.

A.4 Ficheros BFL

Construidos por la correlación de ficheros AFL, un fichero BFL está formado por un conjunto de líneas (en ASCII) cada una de las cuales identifica un flujo bidireccional. En concreto, cada línea (flujo) consta de los siguientes campos, separados por espacios.

Local y remoto están referenidos a la Comunidad Autónoma a la que se refiere el flujo.

VPI/VCI. Circuito ATM en el que se capturaron los paquetes del flujo. Identifica a una Comunidad Autónoma.

Protocolo. Protocolo de nivel de transporte utilizado (según [RFC 1700])

Dirección IP local.

Puerto local. Solo tiene sentido cuando el protocolo es TCP o UDP.

Dirección IP remota.

Puerto remoto. Solo tiene sentido cuando el protocolo es TCP o UDP.

Bytes cursados . Se indican los bytes cursados por el flujo, en ambos sentidos (separados por '/').

Número de paquetes. Paquetes cursados por el flujo, también en ambos sentidos.

Lista de síntomas. Síntomas detectados en el flujo (separados por ';'), también en ambos sentidos. Observar que la lista puede estar vacía.

Veredicto. Categoría de tráfico en la que se ha clasificado el flujo (ACA, COM, LUD o IND). Existirá si se ha realizado clasificación anticipada en *anaflow* (basada en address.cfg).

Pueden agregarse líneas de comentario con #, que son ignorada por los procesos que tratan este tipo de ficheros.

Al final del fichero aparecen tres líneas indicando los instantes de inicio y finalización de la captura (tiempo UNIX), así como el intervalo de tiempo durante el cual se realizó captura efectiva de paquetes (segundos). Estas marcas se muestran por parejas, separadas por '/' sentidos de entrada y salida.

Ejemplo de un fragmento de fichero BFL:

```
212/169 6 192.168.50.246 21 192.168.222.85 2976 0/450 0/5 MAIL=0/5;
212/169 6 192.168.51.10 3218 192.168.43.20 80 0/389 0/2 HTTP=0/1;
212/169 6 192.168.51.136 1093 192.168.22.8 6699 0/5890 0/11  LUD
212/169 6 192.168.53.189 1383 192.168.56.140 80 0/2940 0/20 HTTP=0/5;
212/169 6 192.168.55.122 1124 192.168.230.210 80 0/746 0/4 HTTP=0/1; LUD
212/169 6 192.168.58.5 1129 192.168.17.66 80 0/40 0/1
212/169 6 192.168.58.102 2055 192.168.166.67 80 0/1152 0/24  COM
212/169 6 192.168.59.180 1113 192.168.32.163 80 0/48 0/1
212/169 6 192.168.61.16 1053 192.168.7.131 80 0/239 0/1 HTTP=0/1;
212/169 6 192.168.61.16 1309 192.168.7.131 80 0/781 0/1
212/169 6 192.168.61.16 4637 192.168.7.131 80 0/279 0/2 HTTP=0/1;
init_time=1014804021.226581/1014804032.623769
final_time=1014807593.299606/1014807598.276653
cap_time=89.226756/174.302033
```

A.5 Ficheros VFL

Construidos a partir de los BFL, un fichero VFL está formado por un conjunto de líneas (en ASCII) cada una de las cuales identifica un flujo bidireccional y un veredicto. En concreto, cada línea (flujo) consta de los siguientes campos, separados por espacios.

Local y remoto están referenidos a la Comunidad Autónoma a la que se refiere el flujo.

VPI/VCI. Circuito ATM en el que se capturaron los paquetes del flujo. Identifica a una Comunidad Autónoma.

Protocolo. Protocolo de nivel de transporte utilizado (según [RFC 1700])

Dirección IP local.

Puerto local. Solo tiene sentido cuando el protocolo es TCP o UDP.

Dirección IP remota.

Puerto remoto. Solo tiene sentido cuando el protocolo es TCP o UDP.

Bytes cursados . Se indican los bytes cursados por el flujo, en ambos sentidos (separados por '/').

Número de paquetes. Paquetes cursados por el flujo, también en ambos sentidos.

Lista de síntomas. Síntomas detectados en el flujo (separados por ';'), también en ambos sentidos. Observar que la lista puede estar vacía: en ese caso el veredicto se ha dado en función de las direcciones IP o puertos del flujo.

Veredicto. Categoría de tráfico en la que se ha clasificado el flujo (ACA, COM, LUD o IND).

Pueden agregarse líneas de comentario con #, que son ignorada por los procesos que tratan este tipo de ficheros.

Al final del fichero aparecen tres líneas indicando los instantes de inicio y finalización de la captura (tiempo UNIX), así como el intervalo de tiempo durante el cual se realizó captura efectiva de paquetes (segundos). Estas marcas se muestran por parejas, separadas por '/' sentidos de entrada y salida.

Existen tres subformatos distintos de fichero VFL.

- VFL salida de *vflow*. El descrito hasta ahora.

  Ejemplo:

```
219/169 6 192.168.32.199 1767 192.168.200.2 80 0/1347 0/2 HTTP=0/1; ACA
219/169 6 192.168.33.8 2200 192.168.195.85 3083 2295/0 5/0 HTTP=5/0;SEX=5/0; LUD
219/169 6 192.168.33.8 9000 192.168.307.217 1271 1288/0 6/0 HTTP=2/0; IND
219/169 6 192.168.33.8 80 192.168.488.21 60819 160/0 4/0  ACA
219/169 6 192.168.33.8 2200 192.168.147.47 1073 3096/6912 9/12 HTTP=9/12;MUSICA=0/1;SEX=9/1; LUD
219/169 6 192.168.33.8 80 192.168.62.150 4626 371/0 2/0 HTTP=1/0; ACA
219/169 6 192.168.33.8 80 192.168.156.247 1969 40/2470 1/2 HTTP=0/1; ACA
219/169 6 192.168.33.66 80 192.168.75.52 56634 40/48 1/1  ACA
219/169 6 192.168.33.72 2080 192.168.72.72 80 695/0 4/0 HTTP=1/0;GIF=1/0; ACA
219/169 6 192.168.33.91 1418 192.168.41.2 25 1200/0 30/0  ACA
219/169 6 192.168.33.118 25 192.168.18.10 1639 2840/0 2/0  ACA
init_time=1013461221.913492/1013461202.186793
final_time=1013464800.107253/1013464798.966878
cap_time=436.897369/199.548089
```

- VFL salida de *econv*. Igual que el anterior, pero se añade un campo GROUP al final de todos. Este campo indica el grupo de aplicación en el que se ha clasificado el flujo. Ver 5.4.2 para una explicación más detallada.

  Ejemplo:

```
219/169 6 192.168.32.199 1767 192.168.200.2 80 0/1347 0/2 HTTP=0/1; ACA GROUP=WWW
219/169 6 192.168.33.8 2200 192.168.195.85 3083 2295/0 5/0 HTTP=5/0;SEX=5/0; LUD GROUP=NULL
219/169 6 192.168.33.8 9000 192.168.307.217 1271 1288/0 6/0 HTTP=2/0; IND GROUP=NULL
219/169 6 192.168.33.8 80 192.168.488.21 60819 160/0 4/0  ACA GROUP=WWW
219/169 6 192.168.33.8 2200 192.168.147.47 1073 3096/6912 9/12 HTTP=9/12;MUSICA=0/1;SEX=9/1; LUD GROUP=NULL
219/169 6 192.168.33.8 80 192.168.62.150 4626 371/0 2/0 HTTP=1/0; ACA GROUP=WWW
219/169 6 192.168.33.8 80 192.168.156.247 1969 40/2470 1/2 HTTP=0/1; ACA GROUP=WWW
219/169 6 192.168.33.66 80 192.168.75.52 56634 40/48 1/1  ACA GROUP=WWW
219/169 6 192.168.33.72 2080 192.168.72.72 80 695/0 4/0 HTTP=1/0;GIF=1/0; ACA GROUP=WWW
219/169 6 192.168.33.91 1418 192.168.41.2 25 1200/0 30/0  ACA GROUP=MAIL
219/169 6 192.168.33.118 25 192.168.18.10 1639 2840/0 2/0  ACA GROUP=MAIL
init_time=1013461221.913492/1013461202.186793
final_time=1013464800.107253/1013464798.966878
cap_time=436.897369/199.548089
```

- VFL consolidado, salida de *mflow* cuando consolida este tipo de ficheros. Igual que el primero, pero incluye un nuevo campo antes del VPI/VCI, consistente en un valor entero entre 0 y $m-1$ (siendo m el número de de ficheros consolidados) para indicar la fibra de la que proviene del flujo.

A.6 Ficheros REP

- REP salida de *vflow*. Consiste en un conjunto de líneas (en ASCII), cada una de las cuales muestra los resultados de tráfico para cada una de las Comunidades Autónomas. El primer campo es un acrónimo del nombre de la Comunidad, el segundo es el VPI/VCI que la identifica. Los siguientes campos van de cuatro en cuatro (paquetes de entrada, paquetes de salida, bytes de entrada, bytes de salida), para cada una de las cuatro categorías (ACA, LUD, COM e IND) y el total de todas ellas. Es decir, un total de 20 campos. Se utiliza notación ciéntifica para indicar valores muy grandes, si es necesario.

  Entre estas líneas existe una entrada TOT (cuyo VPI/VCI asociado es */*) que muestra el total de los resultados, es decir, una suma columna a columna de los datos de las Comunidades Autónomas.

  Pueden agregarse líneas de comentario con #, que son ignorada por los procesos que tratan este tipo de ficheros.

  Al final del fichero aparecen tres líneas indicando los instantes de inicio y finalización de la captura (tiempo UNIX), así como el intervalo de tiempo durante el cual se realizó captura efectiva de paquetes (segundos). Estas marcas se muestran por parejas, separadas por '/' sentidos de entrada y salida.

  Ejemplo (algunos saltos de línea son para mejorar la legibilidad):

```
#COMU VPI/VCI IPACA OPACA IBACA OBACA IPLUD OPLUD IBLUD OBLUD IPCOM
    OPCOM IBCOM OBCOM IPIND OPIND IBIND OBIND IPTOT OPTOT IBTOT OBTOT

CAL 242/169 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CAT 252/169 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
EXT 235/169 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CAN 226/169 170620 129416 9.17183e+07 4.81364e+07 40431 32909
   2.58318e+07 9.63524e+06 3776 4227 1.50553e+06 2.0545e+06 6222 4463
   3.56362e+06 1.34214e+06 221049 171015 1.22619e+08 6.11682e+0
CAB 214/69 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

...

TOT */* 1.75845e+06 997664 1.04992e+09 5.20823e+08 385177 261226
   2.04697e+08 1.08171e+08 29485 23980 1.9487e+07 1.47716e+07 86726 68469
```

```
  2.57845e+07 4.22886e+07 2.25984e+06 1.35134e+06 1.29989e+09 6.86054e+08

init_time=968263207.650134/968263200.420072
final_time=968266793.707314/968266799.651422
cap_time=363.355699/292.486748
```

- REP consolidado, salida de *mflow* cuando consolida este tipo de ficheros. Igual que el anterior, salvo que se añade un campo adicional entre el acrónimo de la Comunidad y el VPI/VCI, consistente en un valor entero entre 0 y $m-1$ (siendo m el número de de ficheros consolidados). Es decir, es como concatenar los distintos REP consolidados, añadiendo el nuevo campo para indicar la fibra de origen. La línea de totales contiene igual que antes la suma columna a columna.

  En las líneas de tiempo, aparecen tantas parejas de marcas como ficheros consolidados (es decir, m).

- REP salida de *sumrep*. Se agregan todos los datos referentes a una misma Comunidad Autónoma, aunque provengan de fibras distintas. Igual que el REP salida de *vflow* salvo que desaparece el campo VPI/VCI. La línea de totales contiene igual que antes la suma columna a columna.

  En las líneas de tiempo aparecen tantas parejas de marcas como ficheros consolidados (es decir, m).

Apéndice B

Interfaces de llamada

En este apéndice se incluyen las interfaces de llamada de *shell* que permiten arrancar los programas desarrollados como parte del proyecto fin de carrera. En concreto, se incluye la interfaz de *rm2rs*, *cap2pcap*, *snortflow* y *econv*. Junto a la explicación de los argumentos de llamada se incluye el valor típico (cuando es aplicable).

No se incluye la interfaz de los procesos descritos en el apartado 2.3.1, puesto que son anteriores a la realización de este trabajo. En todo caso, esta información puede encontrarse en [Lago 00].

B.1 *rm2rs*

```
Usage: rm2rs -r rules_MIRA -a address_MIRA -o output_file [-v]
```

Donde:

-r rules_MIRA, fichero de reglas MIRA (rule.cfg).

-a address_MIRA, fichero de direcciones MIRA (address.cfg).

-o output_file, fichero de salida (snort.mira).

-v, modo verboso (opcional).

B.2 *cap2pcap*

```
Usage: cap2pcap -i input_file -o output_file [-a num] &
```

Donde:

-i input_file, directorio de entrada CAP (cap0 o cap1).

-o output_file, fichero de salida PCAP (pcap0 o pcap1).

-a num, número máximo de ficheros en el directorio de salida (opcional, típicamente 50).

B.3 *snortflow*

`Usage: snortflow -i IDIR -r RDIR -c RCFG -a ACFG -s DSNORT -l LDIR [-x] [-n] &`

Donde:

-i IDIR, directorio de entrada PCAP (pcap0 o pcap1).

-r RDIR, directorio de salida AFL (afl0 o afl1).

-c RCFG, fichero de patrones MIRA (rules.cfg).

-a ACFG, fichero de direcciones MIRA (address.cfg).

-s DSNORT, directorio de configuración de Snort (snortconf).

-l LDIR, fichero de salida para Snort (snortlog0 o snortlog1).

-x, no borrar ficheros de entrada tras su procesado (opcional).

-n, modo no bloqueante[1] (opcional).

B.4 *econv*

`Usage: econv -i idir -o odir -v vdir -c cfg [-x] &`

Donde:

-i idir, directorio de entrada VFL (vfl-t).

-o odir, directorio raíz de salida de estadísticas (stat).

-v vdir, directorio de salida VFL (vfl).

-c cfg, fichero de configuración (classports.cfg).

-x, no borrar los ficheros de entrada tras su procesado (opcional).

[1] En el modo no bloqueante, cuando no hay más ficheros en el directorio de entrada, el proceso finaliza, en lugar de esperar a que se le proporcionen más, como es el comportamiento habitual.

Apéndice C

Configuración de seguridad

C.1 Configuración de Snort

A continuación se incluye el fichero snortconf/snort.conf de configuración utilizado por *snortflow*, donde se incluye tanto la configuración de los módulos preprocesadores como la inclusión de las reglas.

```
#--------------------------------------------------
#           SNORTFLOW CONFIGURATION FILE
#       created by Fermín Galán. galan@dit.upm.es
#                 based upon:
#
#   http://www.snort.org     Snort 1.8.4 Ruleset
#     Contact: snort-sigs@lists.sourceforge.net
#--------------------------------------------------
# NOTE:This ruleset only works for 1.8.0 and later
#--------------------------------------------------
# $Id: snort.conf,v 1.77.2.7 2002/03/02 05:33:01 cazz Exp $
#
###################################################
# This file contains a sample snort configuration.
# You can take the following steps to create your
# own custom configuration:
#
#  1) Set the network variables for your network
#  2) Configure preprocessors
#  3) Configure output plugins
#  4) Customize your rule set
#
###################################################
# Step #1: Set the network variables:
#
# You must change the following variables to reflect
# your local network. The variable is currently
# setup for an RFC 1918 address space.
#
# You can specify it explicitly as:
#
# var HOME_NET 10.1.1.0/24
#
# or use global variable $<interfacename>_ADDRESS
```

```
# which will be always initialized to IP address and
# netmask of the network interface which you run
# snort at.
#
# var HOME_NET $eth0_ADDRESS
#
# You can specify lists of IP addresses for HOME_NET
# by separating the IPs with commas like this:
#
# var HOME_NET [10.1.1.0/24,192.168.1.0/24]
#
# MAKE SURE YOU DON'T PLACE ANY SPACES IN YOUR LIST!
#
# or you can specify the variable to be any IP address
# like this:

var HOME_NET any

# Set up the external network addresses as well.
# A good start may be "any"

var EXTERNAL_NET any

# Set up your SMTP servers, or simply configure them
# to HOME_NET

var SMTP $HOME_NET

# Set up your web servers, or simply configure them
# to HOME_NET

var HTTP_SERVERS $HOME_NET

# Set up your sql servers, or simply configure them
# to HOME_NET

var SQL_SERVERS $HOME_NET

# Define the addresses of DNS servers and other hosts

var DNS_SERVERS $HOME_NET

var RULE_PATH /home/mehari/bin/cfg/snortconf

###################################################
# Step #2: Configure preprocessors
#
# General configuration for preprocessors is of
# the form
# preprocessor <name_of_processor>: <configuration_options>

# frag2: IP defragmentation support
# -------------------------------
# This preprocessor performs IP defragmentation.  This plugin will also detect
# people launching fragmentation attacks (usually DoS) against hosts.  No
# arguments loads the default configuration of the preprocessor, which is a
# 60 second timeout and a 4MB fragment buffer.

# The following (comma delimited) options are available for frag2
#    timeout [seconds] - sets the number of [seconds] than an unfinished
#                        fragment will be kept around waiting for completion,
#                        if this time expires the fragment will be flushed
```

```
#    memcap [bytes] - limit frag2 memory usage to [number] bytes
#                      (default:  4194304)

preprocessor frag2

# stream4: stateful inspection/stream reassembly for Snort
#----------------------------------------------------------------------
# Use in concert with the -z [all|est] command line switch to defeat
# stick/snot against TCP rules.  Also performs full TCP stream
# reassembly, stateful inspection of TCP streams, etc.  Can statefully
# detect various portscan types, fingerprinting, ECN, etc.

# stateful inspection directive
# no arguments loads the defaults (timeout 30, memcap 8388608)
# options (options are comma delimited):
#   detect_scans - stream4 will detect stealth portscans and generate alerts
#                  when it sees them when this option is set
#   detect_state_problems - detect TCP state problems, this tends to be very
#                           noisy because there are a lot of crappy ip stack
#                           implementations out there
#   keepstats [machine|binary] - keep session statistics, add "machine" to
#                         get them in a flat format for machine reading, add
#                         "binary" to get them in a unified binary output
#                         format
#   noinspect - turn off stateful inspection only
#   timeout [number] - set the session timeout counter to [number] seconds,
#                      default is 30 seconds
#   memcap [number] - limit stream4 memory usage to [number] bytes
#   log_flushed_streams - if an event is detected on a stream this option will
#                         cause all packets that are stored in the stream4
#                         packet buffers to be flushed to disk.  This only
#                         works when logging in pcap mode!

preprocessor stream4: detect_scans

# tcp stream reassembly directive
# no arguments loads the default configuration
#   Only reassemble the client,
#   Only reassemble the default list of ports (See below),
#   Give alerts for "bad" streams
#
# Available options (comma delimited):
#   clientonly - reassemble traffic for the client side of a connection only
#   serveronly - reassemble traffic for the server side of a connection only
#   both - reassemble both sides of a session
#   noalerts - turn off alerts from the stream reassembly stage of stream4
#   ports [list] - use the space separated list of ports in [list], "all"
#                  will turn on reassembly for all ports, "default" will turn
#                  on reassembly for ports 21, 23, 25, 53, 80, 143, 110, 111
#                  and 513

preprocessor stream4_reassemble

# http_decode: normalize HTTP requests
# ------------------------------------
# http_decode normalizes HTTP requests from remote
# machines by converting any %XX character
# substitutions to their ASCII equivalent. This is
# very useful for doing things like defeating hostile
# attackers trying to stealth themselves from IDSs by
# mixing these substitutions in with the request.
# Specify the port numbers you want it to analyze as arguments.
```

```
# You may also specify -unicode to turn off detection of
# UNICODE directory traversal, etc attacks.  Use -cginull to
# turn off detection of CGI NULL code attacks.

preprocessor http_decode: 80 -unicode -cginull


# unidecode: normalize HTTP/detect UNICODE attacks
# ----------------------------------------------
# Works much the same as http_decode, but does a better
# job of categorizing and identifying UNICODE attacks,
# recommended as a potential replacement for http_decode.

# preprocessor unidecode: 80 -unicode -cginull


# rpc_decode: normalize RPC traffic
# ---------------------------------
# RPC may be sent in alternate encodings besides the usual
# 4-byte encoding that is used by default.  This preprocessor
# normalized RPC traffic in much the same way as the http_decode
# preprocessor.  This plugin takes the ports numbers that RPC
# services are running on as arguments.

preprocessor rpc_decode: 111 32771

# bo: Back Orifice detector
# -------------------------
# Detects Back Orifice traffic on the network.  This preprocessor
# uses the Back Orifice "encryption" algorithm to search for
# traffic conforming to the Back Orifice protocol (not BO2K).
# This preprocessor can take two arguments.  The first is "-nobrute"
# which turns off the plugin's brute forcing routine (brute forces
# the key space of the protocol to find BO traffic).  The second
# argument that can be passed to the routine is a number to use
# as the default key when trying to decrypt the traffic.  The
# default value is 31337 (just like BO).  Be aware that turning on
# the brute forcing option runs the risk of impacting the overall
# performance of Snort, you've been warned...

preprocessor bo: -nobrute

# telnet_decode: Telnet negotiation string normalizer
# ---------------------------------------------------
# This preprocessor "normalizes" telnet negotiation strings from
# telnet and ftp traffic.  It works in much the same way as the
# http_decode preprocessor, searching for traffic that breaks up
# the normal data stream of a protocol and replacing it with
# a normalized representation of that traffic so that the "content"
# pattern matching keyword can work without requiring modifications.
# This preprocessor requires no arguments.

preprocessor telnet_decode

# portscan: detect a variety of portscans
# ---------------------------------------
# portscan preprocessor by Patrick Mullen <p_mullen@linuxrc.net>
# This preprocessor detects UDP packets or TCP SYN packets going to
# four different ports in less than three seconds. "Stealth" TCP
# packets are always detected, regardless of these settings.

preprocessor portscan: $HOME_NET 4 3 portscan.log
```

```
# Use portscan-ignorehosts to ignore TCP SYN and UDP "scans" from
# specific networks or hosts to reduce false alerts. It is typical
# to see many false alerts from DNS servers so you may want to
# add your DNS servers here. You can all multiple hosts/networks
# in a whitespace-delimited list.
#
#preprocessor portscan-ignorehosts: 0.0.0.0

# Spade: the Statistical Packet Anomaly Detection Engine
#-------------------------------------------------------
# READ the README.Spade file before using this plugin!
#
# preprocessor spade: <anom-report-thresh> <state-file>
# <log-file> <prob-mode> <checkpoint-freq>  [-corrscore]
#
# set this to a directory Spade can read and write to
# store its files
#
# var SPADEDIR .
#
# preprocessor spade: -1 $SPADEDIR/spade.rcv $SPADEDIR/log.txt 3 50000
#
# put a list of the networks you are interested in Spade observing packets
# going to here; separate these by spaces
#
# preprocessor spade-homenet: 0.0.0.0/0
#
# this causes Spade to adjust the reporting threshold automatically
# the first argument is the target rate of alerts for normal circumstances
# (0.01 = 1% or you can give it an hourly rate) after the first hour (or
# however long the period is set to in the second argument), the reporting
# threshold given above is ignored you can comment this out to have the
# threshold be static, or try one of the other adapt methods below
# preprocessor spade-adapt3: 0.01 60 168
#
# other possible Spade config lines:
# adapt method #1
#preprocessor spade-adapt: 20 2 0.5
# adapt method #2
#preprocessor spade-adapt2: 0.01 15 4 24 7
# offline threshold learning
#preprocessor spade-threshlearn: 200 24
# periodically report on the anom scores and count of packets seen
#preprocessor spade-survey:  $SPADEDIR/survey.txt 60
# print out known stats about packet feature
#preprocessor spade-stats: entropy uncondprob condprob

# arpspoof
#----------------------------------------
# Experimental ARP detection code from Jeff Nathan, detects ARP attacks,
# unicast ARP requests, and specific ARP mapping monitoring.  To make use
# of this preprocessor you must specify the IP and hardware address of hosts on
# the same layer 2 segment as you.  Specify one host IP MAC combo per line.
# Also takes a "-unicast" option to turn on unicast ARP request detection.

#preprocessor arpspoof
#preprocessor arpspoof_detect_host: 192.168.40.1 f0:0f:00:f0:0f:00


###################################################################
# Step #3: Configure output plugins
```

```
#
# Uncomment and configure the output plugins you decide to use.
# General configuration for output plugins is of the form:
#
# output <name_of_plugin>: <configuration_options>
#
# alert_syslog: log alerts to syslog
# ----------------------------------
# Use one or more syslog facilities as arguments
#
# output alert_syslog: LOG_AUTH LOG_ALERT

# log_tcpdump: log packets in binary tcpdump format
# -------------------------------------------------
# The only argument is the output file name.
#
# output log_tcpdump: snort.log

# database: log to a variety of databases
# ---------------------------------------
# See the README.database file for more information about configuring
# and using this plugin.
#
# output database: log, mysql, user=root password=test dbname=db host=localhost
# output database: alert, postgresql, user=snort dbname=snort
# output database: log, unixodbc, user=snort dbname=snort
# output database: log, mssql, dbname=snort user=snort password=test

# xml: xml logging
# ----------------
# See the README.xml file for more information about configuring
# and using this plugin.
#
# output xml: log, file=/var/log/snortxml

# unified: Snort unified binary format alerting and logging
# -------------------------------------------------------------
# The unified output plugin provides two new formats for logging
# and generating alerts from Snort, the "unified" format.  The
# unified format is a straight binary format for logging data
# out of Snort that is designed to be fast and efficient.  Used
# with barnyard (the new alert/log processor), most of the overhead
# for logging and alerting to various slow storage mechanisms
# such as databases or the network can now be avoided.
#
# Check out the spo_unified.h file for the data formats.
#
# Two arguments are supported.
#    filename - base filename to write to (current time_t is appended)
#    limit    - maximum size of spool file in MB (default: 128)
#
# output alert_unified: filename snort.alert, limit 128
# output log_unified: filename snort.log, limit 128


# trap_snmp: SNMP alerting for Snort
# -------------------------------------------------------------
# Read the README-SNMP file for more information on enabling and using this
# plug-in.
#
#
# The SnmpTrapGenerator outputplugin requires several parameters
```

```
# The parameters depend on the Snmpversion that is used (specified)
# For the SNMPv2c case the paremeters will be as follows
#  alert, <sensorID>, {trap|inform} -v <SnmpVersion> -p <portNumber>
#        <hostName> <community>
#
# For SNMPv2c traps
#
#output trap_snmp: alert, 7, trap -v 2c -p 162  myTrapListener myCommunity
#
# For SNMPv2c informs
#
#output trap_snmp: alert, 7, inform -v 2c -p 162  myTrapListener myCommunity
#
# For SNMPv3 traps with
# security name = snortUser
# security level = authentication and privacy
# authentication parameters :
#          authentication protocol = SHA ,
#          authentication pass phrase = SnortAuthPassword
# privacy (encryption) parameters
#          privacy protocol = DES,
#          privacy pass phrase = SnortPrivPassword
#
#output trap_snmp: alert, 7, trap -v 3 -p 162 -u snortUser -l authPriv -a SHA \
   -A SnortAuthPassword -x DES -X SnortPrivPassword myTrapListener
#For SNMPv3 informs with authentication and encryption
#output trap_snmp: alert, 7, inform -v 3 -p 162 -u snortUser -l authPriv -a SHA \
   -A SnortAuthPassword -x DES -X SnortPrivPassword myTrapListener

# You can optionally define new rule types and associate one or
# more output plugins specifically to that type.
#
# This example will create a type that will log to just tcpdump.
# ruletype suspicious
# {
#   type log
#   output log_tcpdump: suspicious.log
# }
#
# EXAMPLE RULE FOR SUSPICIOUS RULETYPE:
# suspicious $HOME_NET any -> $HOME_NET 6667 (msg:"Internal IRC Server";)
#
# This example will create a rule type that will log to syslog
# and a mysql database.
# ruletype redalert
# {
#   type alert
#   output alert_syslog: LOG_AUTH LOG_ALERT
#   output database: log, mysql, user=snort dbname=snort host=localhost
# }
#
# EXAMPLE RULE FOR REDALERT RULETYPE
# redalert $HOME_NET any -> $EXTERNAL_NET 31337 (msg:"Someone is being LEET"; \
#   flags:A+;)

#
# Include classification & priority settings
#

include classification.config
```

```
###################################################################
# Step #4: Customize your rule set
#
# Up to date snort rules are available at http://www.snort.org
#
# The snort web site has documentation about how to write your own
# custom snort rules.
#
# The rules included with this distribution generate alerts based on
# on suspicious activity. Depending on your network environment, your
# security policies, and what you consider to be suspicious, some of
# these rules may either generate false positives ore may be detecting
# activity you consider to be acceptable; therefore, you are
# encouraged to comment out rules that are not applicable in your
# environment.
#
# Note that using all of the rules at the same time may lead to
# serious packet loss on slower machines. YMMV, use with caution,
# standard disclaimers apply. :)
#
# The following individuals contributed many of rules in this
# distribution.
#
# Credits:
#   Ron Gula <rgula@securitywizards.com> of Network Security Wizards
#   Max Vision <vision@whitehats.com>
#   Martin Markgraf <martin@mail.du.gtn.com>
#   Fyodor Yarochkin <fygrave@tigerteam.net>
#   Nick Rogness <nick@rapidnet.com>
#   Jim Forster <jforster@rapidnet.com>
#   Scott McIntyre <scott@whoi.edu>
#   Tom Vandepoel <Tom.Vandepoel@ubizen.com>
#   Brian Caswell <bmc@snort.org>
#   Zeno <admin@cgisecurity.com>
#   Ryan Russell <ryan@securityfocus.com>
#
#=========================================
# Include all relevant rulesets here
#
# shellcode, policy, info, backdoor, and virus rulesets are
# disabled by default.  These require tuning and maintance.
# Please read the included specific file for more information.
#=========================================

# Incluir reglas de MIRA
include $RULE_PATH/snort.mira

# Puede elegirse una configuración de reglas de seguridad basada en
# ficheros por tipos de ataque o por prioridad.
#
# Actualmente se usa una configuración basada en las reglas de
# prioridad 1.

# 1) Por tipos de ataque. A continuación se encuentra la lista
# de todos los ficheros incluidos en la configuración de Snort.
# Activar (descomentar) aquellos que se desee utilizar.

#include $RULE_PATH/bad-traffic.rules
#include $RULE_PATH/exploit.rules
#include $RULE_PATH/scan.rules
#include $RULE_PATH/finger.rules
#include $RULE_PATH/ftp.rules
```

```
#include $RULE_PATH/telnet.rules
#include $RULE_PATH/smtp.rules
#include $RULE_PATH/rpc.rules
#include $RULE_PATH/rservices.rules
#include $RULE_PATH/dos.rules
#include $RULE_PATH/ddos.rules
#include $RULE_PATH/dns.rules
#include $RULE_PATH/tftp.rules
#include $RULE_PATH/web-cgi.rules
#include $RULE_PATH/web-coldfusion.rules
#include $RULE_PATH/web-iis.rules
#include $RULE_PATH/web-frontpage.rules
#include $RULE_PATH/web-misc.rules
#include $RULE_PATH/web-attacks.rules
#include $RULE_PATH/sql.rules
#include $RULE_PATH/x11.rules
#include $RULE_PATH/icmp.rules
#include $RULE_PATH/netbios.rules
#include $RULE_PATH/misc.rules
#include $RULE_PATH/attack-responses.rules
#include $RULE_PATH/backdoor.rules
#include $RULE_PATH/shellcode.rules
#include $RULE_PATH/policy.rules
#include $RULE_PATH/porn.rules
#include $RULE_PATH/info.rules
#include $RULE_PATH/icmp-info.rules
#include $RULE_PATH/virus.rules
#include $RULE_PATH/experimental.rules
#include $RULE_PATH/local.rules

# 2) Por prioridad. Se especifican tres ficheros, cada uno
# con el conjunto de reglas de los ficheros anteriores que tienen
# un determinado nivel de prioridad. Activar aquellos que se
# desee utilizar.

include $RULE_PATH/priority-1.rules
#include $RULE_PATH/priority-2.rules
#include $RULE_PATH/priority-3.rules
```

C.2 *Scripts* de consolidación de alertas

A continuación se incluye el código de los *scripts* de comandos consolidate_logs.sh y sum_logs.sh utilizados para la consolidación de alertas de seguridad y generación de informes HTML.

C.2.1 *consolidate_logs.sh*

```
#!/bin/sh
#
# Script de mantenimiento de los registros de snort consolidate_logs.sh
# Fermín Galán. galan@dit.upm.es

# Variables

DAT=/home/mehari/data
```

```
BIN=/home/mehari/bin
HTML=/var/www/html/snort/

# 1. Agregar información de todas las maquinas
ssh -n mira0 ${BIN}/sum_logs.sh 0
ssh -n mira1 ${BIN}/sum_logs.sh 1
ssh -n mira2 ${BIN}/sum_logs.sh 2

# 2. Consolidar los registros de paquetes de las tres máquinas
timestamp1=$(date "+%y-%m-%d-%T")
timestamp2=$(date "+%y%m%d%H%M%S")

mergecap -w ${DAT}/slog/SLOG$timestamp2 ${DAT}/slog/sumalerts-1 \
   ${DAT}/slog/sumalerts-2 ${DAT}/slog/sumalerts-3

# 3. Comprimir registros de paquetes
gzip ${DAT}/slog/SLOG$timestamp2

# 4. Crear informe HTML
mkdir $HTML/$timestamp1
${BIN}/snortsnarf.pl -d $HTML/$timestamp1 -rulesfile ${CFG}/snortconf/snort.conf \
   -rulesdir ${CFG}/snortconf ${DAT}/slog/alert-0-0 ${DAT}/slog/alert-0-1 \
   ${DAT}/slog/alert-1-0 ${DAT}/slog/alert-1-1 ${DAT}/slog/alert-2-0 ${DAT}/slog/alert-2-1 \
   ${DAT}/slog/portscan-0-0.log ${DAT}/slog/portscan-0-1.log ${DAT}/slog/portscan-1-0.log \
   ${DAT}/slog/portscan-1-1.log ${DAT}/slog/portscan-2-0.log ${DAT}/slog/portscan-2-1.log

# 5. Limpiar
rm ${DAT}/slog/sumalerts-*
rm ${DAT}/slog/alert-*
rm ${DAT}/slog/portscan-*
```

C.2.2 *sum_logs.sh*

```
#!/bin/sh
#
# Script de consolidación de los registros de Snort en cada máquina: sum_logs.sh
# Fermín Galán. galan@dit.upm.es

# Argumentos:
#      $1, máquina en la que se ejecuta el script

# Variables

DAT=/home/mehari/data
BIN=/home/mehari/bin
CFG=/home/mehari/bin/cfg

# 1. Detener procesos snortflow
/bin/kill -9 $(/bin/pidof snortflow)

# 2. Mover las alertas y registros de paquete a los directorios de trabajo
mv ${DAT}/snortlog0/alert* ${DAT}/rotatelog0
mv ${DAT}/snortlog0/portscan.log ${DAT}/rotatelog0
mv ${DAT}/snortlog1/alert* ${DAT}/rotatelog1
mv ${DAT}/snortlog1/portscan.log ${DAT}/rotatelog1

# 3. Rearrancar snortflow (como en mira-ana)
rm ${DAT}/snortlog0/*.clog
rm ${DAT}/snortlog1/*.clog
```

```
${BIN}/snortflow -i ${DAT}/pcap0 -r ${DAT}/afl0 -c ${CFG}/rule.cfg \
   -a ${CFG}/address.cfg -l ${DAT}/snortlog0 -s ${CFG}/snortconf &
${BIN}/snortflow -i ${DAT}/pcap1 -r ${DAT}/afl1 -c ${CFG}/rule.cfg \
   -a ${CFG}/address.cfg -l ${DAT}/snortlog1 -s ${CFG}/snortconf &

# 4. Agregar el conjunto de paquetes registrados
mergecap -w ${DAT}/slog/sumalerts-$1 ${DAT}/rotatelog0/alert-* ${DAT}/rotatelog1/alert-*

# 5. Mover los ficheros de alertas (alert y portscan.log) al directorio compartido
mv ${DAT}/rotatelog0/alert ${DAT}/slog/alert-$1-0
mv ${DAT}/rotatelog0/portscan.log ${DAT}/slog/portscan-$1-0.log
mv ${DAT}/rotatelog1/alert ${DAT}/slog/alert-$1-1
mv ${DAT}/rotatelog1/portscan.log ${DAT}/slog/portscan-$1-1.log

# 6. Limpiar
rm ${DAT}/rotatelog0/*
rm ${DAT}/rotatelog1/*
```

Apéndice D

Algoritmo de Boyer-Moore

El algoritmo de Boyer-Moore (utilizado, entre otros sitios, en el motor de búsqueda de Snort) sirve para realizar búsquedas de patrones de forma bastante rápida, en comparación con otros algoritmos. Vamos a describirlo a continuación.

Consideremos que buscamos un patrón de longitud m en un área de datos de longitud n. Sean $b_1, b_2, \ldots, b_m$ los elementos del patrón y $a_1, a_2, \ldots, a_n$ los del área de datos. Asumimos que $n \geq m$ (sino, al ser más grande el patrón que los datos, estamos ante un caso trivial: la búsqueda falla).

Para realizar la búsqueda, se empieza alineando el patrón con $a_1, a_2, \ldots, a_m$. Posteriormente, se realiza la comparación de los distintos elementos de derecha a izquierda. Es decir, $b_m = a_m$, luego $b_{m-1} = a_{m-1}$, ..., hasta $b_1 = a_1$. Si alguna de estas comparaciones no se satisface, se desplaza el patrón un valor d y se repite la comparación, hasta encontrar una coincidencia completa o finalizar el área de datos (en cuyo caso la búsqueda fracasa).

A diferencia del algoritmo de fuerza bruta, donde $d = 1$ siempre, el algoritmo de Boyer-Moore utiliza heurísticos para evitar realizar comparaciones innecesarias. Consideremos que, en un momento dado, el patrón está desplazado una cantidad D (es decir, b_1 está alineado con a_{D+1}, b_2 con a_{D+2}, ... y b_m con a_{D+m}). Vamos a describir la forma en la que se calcula d.

Sea $c = a_{D+i}$ en caracter donde se produce el fallo de coincidencia. Puesto que la comparación se realiza de derecha a izquierda, se cumple que $\forall j, i < j \leq m$, $b_j = a_{D+j}$. De esta forma, el patrón queda divido en dos partes: la derecha ($j > i$), con coincidencia y que ignoraremos y la izquierda ($j \leq i$), donde aplicamos los heurísticos para determinar el desplazamiento a aplicar. De hecho, si el fallo de coincidencia se produce en el último caracter del patrón ($i = m$), lo cual ocurre bastante habitualmente, solo tenemos parte izquierda.

Consideramos el conjunto de índices de carácteres de la parte izquierda del patrón que coinciden con c. Formalmente[1]:

$$M(c, i) = \{1 \leq j < i \mid b_j = c\}$$

Consideremos dos posibles casos:

- $M = \Phi$. No existe c dentro de la parte izquierda. Podemos desplazar el patrón un número de elementos igual a su longitud, es decir $d = i$.

- $M \neq \Phi$. Existe al menos un c dentro de la parte izquierda. Desplazamos el patrón un número de elementos tal que queden alineados el elemento extremo coindicente del patrón con el c del área de datos. Formalmente, el desplazamiento es $d = i - \max M$.

Observar que el caso mejor del algoritmo es el primero, especialmente si $i = m$, ya que el desplazamiento es máximo.

En definitiva, el desplazamiento puede formalizarse como una función f que puede calcularse antes de comenzar la búsqueda, ya que solo depende del patrón. En concreto esta *función desplazamiento* es:

$$f(c, i) = \begin{cases} i & \text{si } M(c, i) = \Phi \\ i - \max M & \text{si } M(c, i) \neq \Phi \end{cases}$$

Para comprender mejor el funcionamiento del algoritmo, la figura D.1 ilustra como se realizan los desplazamientos. Se busca el patrón "lata" en la frase "sistema plataforma". La función de desplazamiento es (c representa cualquier carácter distinto de los que forman parte del patrón):

$i = 4$	$i = 3$	$i = 2$	$i = 1$
$f(\text{'l'}, 4) = 3$	$f(\text{'l'}, 3) = 2$	$f(\text{'l'}, 2) = 1$	$f(\text{'l'}, 1) = 1$
$f(\text{'a'}, 4) = 2$	$f(\text{'a'}, 3) = 1$	$f(\text{'a'}, 2) = 2$	$f(\text{'a'}, 1) = 1$
$f(\text{'t'}, 4) = 1$	$f(\text{'t'}, 3) = 3$	$f(\text{'t'}, 2) = 2$	$f(\text{'t'}, 1) = 1$
$f(c, 4) = 4$	$f(c, 3) = 3$	$f(c, 2) = 2$	$f(c, 1) = 1$

Compárese en este caso las cuatro comparaciones que requiere el algoritmo de Boyer-Moore con las 10 que requeriría un algoritmo de fuerza bruta en las mismas circustancias.

En el caso peor, el desplazamiento es siempre 1 y el algoritmo de Boyer-Moore es equivalente al de fuerza bruta. De todas formas, esto rara vez ocurre y el algoritmo suele ser bastante rápido, tanto más cuando más largo es el

[1]Observar que no es necesario considerar $j = i$ dentro del recorrido del conjunto, ya que $b_i \neq c$ por definición.

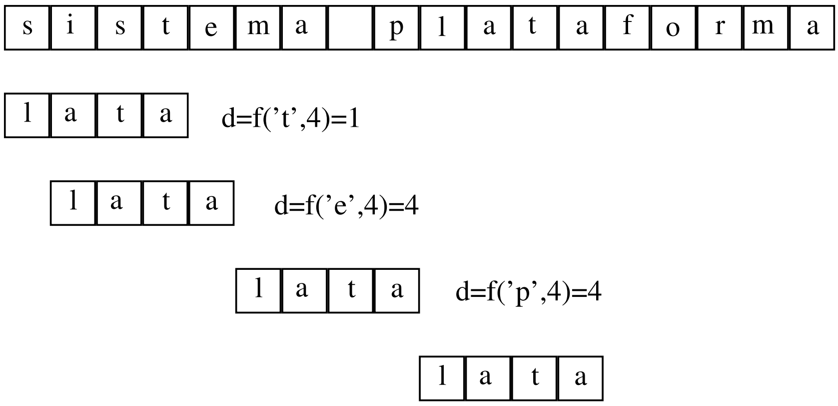


Figura D.1: Ejemplo de funcionamiento del algoritmo de Boyer-Moore.

patrón. Formalmente, el algoritmo es de orden $O(n/m)$ en el mejor de los casos y $O(nm)$ (como fuerza bruta) en el peor.

En este apéndice hemos descrito los fundamentos del algoritmo con algo de detalle, pero sin entrar en aspectos concretos de la implementación. Para encontrar información más detallada puede consultarse [Boyer+ 77].

Apéndice E

Glosario

AAL5 ATM Adaptation Layer Number 5.

AAC Advanced Audio Coding.

ADSL Asymmetric Digital Subscriber Line.

ARP Address Resolution Protocol.

ASCII American Standard Code for Information Interchange.

ATM Asynchronous Transfer Mode.

BMP BitMap Picture.

bps Bits por segundo.

BSD Berkeley Software Distribution.

CAIDA Cooperative Association for Internet Data Analysis.

CASTBA CAlidad de los Servicios Telemáticos provistos por la red académica de Banda Ancha.

CBR Constant Bitrate.

CD Compact Disc.

CERT Computer Emergency Response Team.

CGI Common Gateway Interface.

CICYT Comisión Interministerial de Ciencia y Tecnología (en la actualidad Ministerio de Ciencia y Tecnología).

CLS ConnectionLess Server.

CPU Central Process Unit.

CRC Ciclic Redundance Code.

CS Convergence Sublayer.

CSIC Consejo Superior de Investigaciones Científicas.

CVS Concurrent Version System.

dB Decibelio.

DDoS Distributed Denial of Service.

DMA Direct Memory Access.

DNS Domain Name Service.

DoS Denial of Service.

DTD Document Type Definition.

FDDI Fiber Distributed Data Interface.

FEDER Fondo Europeo de Desarrollo Regional.

FTP File Transfer Protocol.

GMT Global Meridiam Time.

GNU GNU is Not UNIX.

Gpbs Gigabits por segundo (1000 Mbps)

GPRS General Packet Radio Service.

HDLC High-Level Data Link Control.

HFMP HTML Form Processing Modules.

HTML HyperText Markup Language.

HTTP Hypertext Transfer Protocol.

HTTPS HTTP Secure.

Hz Hertzio.

ICMP Internet Control Management Protocol.

IDE Integrated Drive Electronics.

IEC International Electrical Commission.

IEEE Institute of Electrical and Electronics Engineers.

IETF Internet Engineering Task Force.

IGMP Internet Group Management Protocol.

IMAP Internet Message Access Protocol.

IP Internet Protocol.

IRC Internet Relay Chat.

ISO International Standards Organization.

ISP Internet Service Provider.

ITU International Telecomunication Union.

ITU-T ITU Telecomunications Standardization Sector.

Kbps Kilobits por segundo (1000 bps).

kHz Kilohertzio (1000 Hz).

LLC Logical Link Control.

MEHARI MEcanismos y Herramientas para el Ánalisis del uso de servicios Internet aplicado a RedIris.

MHz Megahertzio (1000 kHz).

MIME Multipurpose Internet Mail Extensions.

MIRA Metodologías para la Inspección de Redes Avanzadas.

MP3 MPEG Layer III.

MPEG Motion Pictures Expert Group.

Mpbs Megabits por segundo (1000 Kbps).

MTU Maximun Transfer Unit.

NFS Network File System.

NIC Network Interface Card.

NIDS Network Intrusion Detection System.

NMS Network Management Station.

NTP Network Time Protocol.

P2P Peer to Peer.

PC Personal Computer.

PCI Peripheral Component Interconnect.

PDU Protocol Data Unit.

PHP Hypertext Preprocessor.

POP Post Office Protocol.

POS Packet Over Sonet.

PPP Point to Point Protocol.

QoS Quality of Service.

RAM Random Access Memory.

RFC Request For Coments.

RPC Remote Procedure Call.

SAR Segmentation And Reassembly.

SCSI Small Computer Systems Interface.

SISR SnortSnarf Incidence Storage and Reporting.

SLIP Serial Line Internet Protocol.

SMB Server Message Block.

SMS Short Message Service.

SMTP Simple Mail Transfer Protocol.

SNAP SubNetwork Access Protocol.

SNML SNort Markup Language.

SNMP Simple Network Management Protocol.

SPADE Statistical Packet Anomaly Detection Engine.

SQL Standard Query Language.

SSA Subsistema de análisis (MIRA).

SSC Subsistema de captura (MIRA).

SSP Subsistema de preprocesado (MIRA).

TCP Transmission Control Protocol.

TFTP Trivial File Transfer Protocol.

TTL Time To Live.

UDP User Datagram Protocol.

UML Unified Modeling Language.

UNIX UNiplexed Information and Computing System.

URI Uniform Resource Identifier.

URL Uniform Resource Locator.

VBR Variable Bitrate.

VCI Virtual Circuit Identifier.

VPI Virtual Path Identifier.

WAV Wave Sound Format.

WMA Windows Media Audio.

WWW World Wide Web.

XML eXtended Markup Language.

XMMS eXtended MultiMedia System.

Referencias

[Ahuja 96] Vijay Ahuja. *Network and Internet Security.* AP Professional, 1996. ISBN: 0-12-045595-1

[Álvarez+ 98] M. Álvarez-Campana, A. Azcorra, J. Berrocal, J. Domingo Pascual, D. Larrabeiti, X. Martínez, J. I. Moreno, J. R. Pérez y J. Solé Pareta. *CASTBA: Medidas de tráfico sobre la Red Académica Española de Banda Ancha.* Telecom I+D, Octubre, 1998.

[Álvarez+ 99a] M. Álvarez-Campana, A. Azcorra, J. Berrocal, J. R. Pérez, E. Vázquez. *Internet Traffic Measurements over the Spanish R&D IP/ATM Network Backbone.* 7th IFIP Workshop on Performance Modelling and Evaluation of ATM Networks, Antwerp, Bélgica, Junio 1999.

[Álvarez+ 99b] M. Álvarez-Campana, A. Azcorra, J. Berrocal, A. B. García, J. R. Pérez. *MEHARI: An IP/ATM Traffic Analysis Platform based on Configurable Patterns.* 6th HP Openview University Association Workshop (HPOVUA 99), Bolonia, Italia, 13-15 Junio 1999.

[Azcorra 99] A. Azcorra. *MIRA: Metodologías para la Inspección de tráfico en Redes Avanzadas.* Programa Nacional de Investigación y Desarrollo: Memoria del proyecto, 1999.

[Azcorra+ 99] A. Azcorra, J. Solé Pareta, J. Domingo Pascual y M. Álvarez-Campana. *Análisis del uso de los servicios Internet a partir del sistema MEHARI: Prueba de campo sobre RedIris.* Seminario del Programa Nacional de Aplicaciones y Servicios Telemáticos, SPAST 99, Pamplona, España, Diciembre, 1999.

[AudioCatalyst] *AudioCatalyst.* URL: http://www.xingtech.com/mp3/audiocatalyst/.

[Boyer+ 77] R. S. Boyer y J. S. Moore. *A fast string searching algorithm*. Communications of the ACM, volumen 20, número 10, páginas 762-772, 1977.

[Cheswick+ 94] W. R. Cheswick y S. M. Bellovin. *Firewalls and Internet Security*. Addison Wesley, Noviembre 1994. ISBN: 0-201-63357-4.

[CERT] *CERT Coordination Center*. URL: http://www.cert.org/.

[CoralReef] *CoralReef Software Suite*. URL: http://www.caida.org/tools/measurement/coralreef/.

[Davies+ 94] D. W. Davies y W. L. Prices. *Security for Computer Networks*. John Wiley and Sons, Febrero, 1994. ISBN: 0-471-92137-8.

[DIT 96] DIT-UPM. *Supervisión de calidad de servicios telemáticos sobre la Red Académica de Banda Ancha*. Programa Nacional de Investigación y Desarrollo: Propuesta de acción especial CASTBA, Noviembre, 1996.

[DIT 97] DIT-UPM. *Supervisión de calidad de servicios telemáticos sobre la Red Académica de Banda Ancha*. Programa Nacional de Investigación y Desarrollo: Propuesta de acción especial MEHARI, Septiembre, 1997.

[DIT 98a] DIT-UPM. *Proyecto MEHARI-M. Informe de progreso*. Programa Nacional de Investigación y Desarrollo, Octubre 1998.

[DIT 98b] DIT-UPM. *Proyecto MEHARI-M. Informe final*. Programa Nacional de Investigación y Desarrollo, Octubre 1998.

[divX] *DivX Frequently Asked Questions*. URL: http://www.divx.com/support/faq.php.

[McDysan+ 99] D. McDysan y D. Spohn. *ATM Theory and Applications*. McGraw Hill, 1999. ISBN: 0-07-045346-2.

[Echeverría 01] Javier Echeverría. *Del campus a la red: las universidades en el espacio electrónico*. Jornadas Técnicas RedIris 2001, Pamplona, 2001. Videoconferencia accesible en URL: http://www.rediris.es/jt/jt2001/archivo-jt.es.html.

[ELMUNDO 00] *Un juez ordena el cierre temporal de Napster*. EL MUNDO, Diario del Navegante 2.0, 27 de julio de 2000, URL: http://www.elmundo.es/navegante/2000/07/27/napster.html.

[Ethereal] *Ethereal*. URL: http://www.ethereal.org/.

[Evans 01] B. Evans. *Peer-to-peer Networking*, 2001. URL: http://rr.sans.org/threats/peer2.php.

[Filetopia] *Filetopia. File sharing tool with public key encryption*. URL: http://www.filetopia.org/.

[Forcht 94] K. A. Forcht. *Computer Security Management*. Boyd and Fraser Publishing Company, 1994. ISBN: 0-87835-881-1.

[Fragroute] *Fragroute*. URL: http://www.monkey.org/~dugsong/fragroute/.

[GNU 89] R. Stallman. *GNU General Public Licence*. Free Software Foundation, 1989.

[Guardian] *Guardian Active Response for Snort*. URL: http://www.chaotic.org/guardian/.

[ID3] *ID3v2 The audience is informed*. URL: http://www.id3.org/.

[infoAn] *infoAnarchy Resources*. URL: http://www.infoanarchy.org/special/resources.

[ISO 11172] MPEG Working Group. ISO/IEC-11172-3 *Coding of Moving Pictures and Associated Audio for Digital Storage Media at Up To Abouts 1.5 Mbit/s*. ISO, 1993.

[ISO 138138] MPEG Working Group. ISO/IEC-138138-3 *Generic Coding of Moving Pictures and Associated Audio*. ISO, Noviembre, 1994.

[ITU 93] ITU-T. *Recommendation I.211. B-ISDN Service Aspects*. Marzo, 1993.

[Lago 00] D. Lago San León. *Desarrollo de una Plataforma de Monitorización y Análisis para Redes IP/ATM*. Proyecto Fin de Carrera, Universidad Politécnica de Madrid, Escuela Técnica Superior de Ingenieros de Telecomunicación, 2000.

[Lizcano+ 99] P. J. Lizcano, A. Azcorra, J. Solé Pareta, J. Domingo Pascual, M. Álvarez-Campana. *MEHARI: A System for Analysing the Use of Internet Services.* Computer Networks and ISDN Systems. ISSN: 0169-7552 31(10), Noviembre, 1999.

[Menéndez 00] J. M. Menéndez García y F. J. Casajús Quirós. *Tecnologías de audio y vídeo.* Universidad Politécnica de Madrid, Escuela Técnica Superior de Ingenieros de Telecomunicación, Publicaciones, 2000. ISBN: 84-7402-288-6.

[mp3_check] *mp3_check.* http://freshmeat.net/projects/mp3_check/?topic_id=123.

[Napigator] *Napigator.* URL: http://www.napigator.com/.

[Napster] *Napster Protocol Specification.* URL: http://opennap.sourceforge.net/napster.txt.

[Pérez98] J. R. Pérez, M. Álvarez-Campana, A. Azcorra, D. Larrabeiti, J. I. Moreno. *CASTBA: Internet Traffic Measurements over the Spanish R&D ATM Network.* 5th HP Openview University Association Workshop (HPOVUA 98), Rennes, Francia, Abril, 1998.

[RedIris] *RedIris. Servicio de Red Sideral.* URL: http://www.rediris.es/red/.

[RFC 793] *Transmission Control Protocol.* IETF, RFC 793, Septiembre, 1981.

[RFC 894] C. Horning. *A Standard for the Transmission of IP Datagrams over Ethernet Networks.* IETF, RFC 894, Abril, 1984.

[RFC 1483] J. Heinanen. *Multiprotocol Encapsulation over ATM Adaptation Layer 5.* IETF, RFC 1483, Julio, 1993.

[RFC 1577] M. Laubach. *Classical IP and ARP over ATM.* IETF, RFC 1577, Enero, 1994.

[RFC 1626] R. Atkinson. *Default IP MTU for use over ATM.* IETF, RFC 1626, Mayo 1994.

[RFC 1700] J. Reynolds y J. Postel. *Assigned Numbers.* IETF, RFC 1700, Octubre, 1994.

[Roesch 99] M. Roesch. *Snort: Lightweight Intrusion Detection for Networks.* LISA XIII, págs. 101-110, Noviembre, 1999.

[Roesch 02] M. Roesch. *Snort Users Manual. Snort Release 1.8.4.* Marzo, 2002.

[Romeral 01] R. Romeral Ortega. *Mejora de una Plataforma Avanzada de Inspección de Tráfico.* Proyecto Fin de Carrera, Universidad Carlos III de Madrid, Escuela Politécnica Superior, Febrero, 2001.

[Romeral+ 02] R. Romeral Ortega, A. García-Martínez, A. B. García, A. Azcorra, M. Álvarez-Campana. *MIRA: A Distributed and Scalable WAN/LAN Real-time Measurement Platform.* Second International Workshop on Internet Charging and QoS Technology (ICQT'02). Zürich, Suiza, Octubre, 2002[1].

[Shoutcast] *SHOUTCast MPEG Layer 3 Audio Streaming Technology.* URL: http://www.shoutcast.com/.

[Snort] *Snort. The Open Source Network Intrusion Detection System.* URL: http://www.snort.org/.

[SnortSnarf] *SnortSnarf.* URL: http://www.silicondefense.com/software/snortsnarf/index.htm.

[Snot] *Snot.* URL: http://www.sec33.com/sniph/snot-0.92a-README.txt

[Spade] *Statistical Packet Anomaly Detection Engine.* http://www.silicondefense.com/software/spice/index.htm

[Stallings 95] W. Stallings. *Networks and Internetwork Security Principles and Practice.* Prentice Hall, 1995. ISBN: 0-02-415483-0.

[Stevens 94] W. R. Stevens. *TCP/IP Ilustrated. Volumen 1: The Protocols.* Addison Wesley, 1994. ISBN: 0-201-63346-9.

[Stick] *Stick.* URL: http://www.eurocompton.net/stick/projects8.html

[tcpdump] *TCPDUMP public repository.* URL: http://www.tcpdump.org/.

[1] En evaluación en el momento de redactar esta memoria

[Veciana+ 00] C. Veciana, J. Solé Pareta, S. Sales, J. Domingo Pascual, A. Azcorra, A. García-Martínez, M. Álvarez-Campana, A. B. García, P. A. Aranda Gutiérrez. *MIRA: Software para el análisis de tráfico IP sobre ATM.* ISBN 84-607-1397-0, X Jornadas TELECOM I+D, Madrid - Barcelona - Valencia, España, 22 y 23 de Noviembre, 2000.

[Winamp] *Winamp.* URL: http://www.winamp.com/.

[XMMS] *X MultiMedia System.* URL: http://www.xmms.org/.

Índice Alfabético

Parte II

Planos

Parte III

Pliego de condiciones

1 Introducción

El objetivo del presente proyecto es el desarrollo mejorado de una plataforma de captura y análisis de tráfico, así como la generación de un conjunto de informes derivados del trabajo realizado y del análisis de medidas durante el periodo de realización y pruebas del sistema.

2 Entregables

- Plataforma de captura y monitorización de tráfico. La plataforma consta tanto de los elementos hardware que realizan la captura y procesamiento como del software que se ejecuta sobre los mismos y que incorporará las funcionalidades avanzadas de las que es objeto este proyecto.

- Informes derivados del desarrollo. Se considera a este efecto la realización de informes periódicos de dos tipos: informes técnicos (documentando la realización de actividades durante el proyecto) e informes de medida (con datos estadísticos sobre el tráfico analizado por la plataforma en el periodo de realización y pruebas).

3 Condiciones de desarrollo

3.1 Recursos hardware

- Equipos de captura (tres), para la realización de la captura de tráfico. Características técnicas: PC con microprocesador Intel Pentium a 650 MHz, 256 Mbytes de memoria RAM, disco duro IDE de 8 Gbytes, tarjeta Fast Ethernet 10/100Mbps, dos tarjetas ATM NIC PCI SC-155, tipo Fore PCA-200EPC 256KB y dos divisores de fibra pasivos SC conectados a los puertos de Rx.

- Equipos de análisis (tres), para la realización del análisis y consolidación del tráfico. Características técnicas: PC con microprocesador Intel

Pentium a 650 MHz, 256 Mbytes de memoria RAM, disco duro SCSI 8 Gbytes, dos tarjetas Fast Ethernet 10/100Mbps.

- Equipo de desarrollo, para la gestión remota de los equipos de captura y análisis, así como el desarrollo de pruebas de software y la redacción de documentación. Características técnicas: PC con microprocesador Intel Pentium a 350 MHz, 128 Mbytes de memoria RAM, disco duro IDE de 8 Gbytes, tarjeta Fast Ethernet 10/100Mbps.

- Impresora láser, para la generación de documentación.

3.2 Recursos software

- Sistema operativo FreeBSD 2.2.5, utilizado en los equipos de captura.

- Sistema operativo Debian GNU/Linux 2.2.14, utilizado en los equipos de análisis.

- Herramientas de desarrollo para entornos UNIX (compiladores, editores, etc.), que acompañan a estos sistemas operativos.

- Sistema operativo Windows 2000, utilizado en el equipo de desarrollo.

- Licencia Microsoft Office, para la elaboración de documentación.

Parte IV

Presupuesto

1 Introducción

El presupuesto consta de los siguientes conceptos:

- Presupuesto de ejecución material.
- Gastos generales y Beneficio industrial.
- Honorarios por la redacción y dirección del proyecto.
- Presupuesto total.

Los dos primeros conceptos (Presupuesto de ejecución material y Gastos generales y Beneficio industrial) constituyen el Presupuesto de ejecución por contrata, que junto con los Honorarios por la redacción y dirección del proyecto constituyen el Presupuesto total.

Todos las cantidades se encuentran expresadas en euros.

2 Presupuesto de ejecución material

El Presupuesto de ejecución material consta de Costes de recursos materiales y de Costes de mano de obra, no incluyendo los Honorarios de dirección del proyecto, que se considerarán aparte.

2.1 Descomposición en tareas

El proyecto consta de las siguientes tareas:

- Tarea 1:

  Objetivo: estudio de la plataforma de captura y análisis base del desarrollo.

  Duración: 2 meses.

  Esfuerzo: ingeniero superior, 1 h-m.

Resultado: adquisición de los conocimientos necesarios sobre la plataforma preexistente para acometer los desarrollos de los que son objeto las demás tareas.

- Tarea 2:

  Objetivo: desarrollo de mecanismos avanzados de detección de tráfico lúdico.

  Duración: 4 meses.

  Esfuerzo: ingeniero superior, 0.25 h-m.

  Resultado: análisis diseño e implantación de técnicas para la detección de tráfico lúdico en la plataforma de captura y monitorización.

- Tarea 3:

  Objetivo: desarrollo de funciones de seguridad.

  Duración: 6 meses.

  Esfuerzo: ingeniero superior, 0.25 h-m.

  Resultado: análisis diseño e implantación de funciones para la detección de ataques de seguridad en la plataforma de captura y monitorización.

- Tarea 4:

  Objetivo: desarrollo de un módulo de estadísticas convencionales.

  Duración: 2 meses.

  Esfuerzo: ingeniero superior, 0.25 h-m.

  Resultado: análisis diseño e implantación de un módulo de generación de estadísticas convencionales.

- Tarea 5:

  Objetivo: operación y gestión remota de la plataforma.

  Duración: 6 meses.

  Esfuerzo: ingeniero superior, 0.25 h-m.

  Resultado: realización de la gestión continuada de la plataforma, asegurando el correcto funcionamiento de los equipos y los procesos.

- Tarea 6:

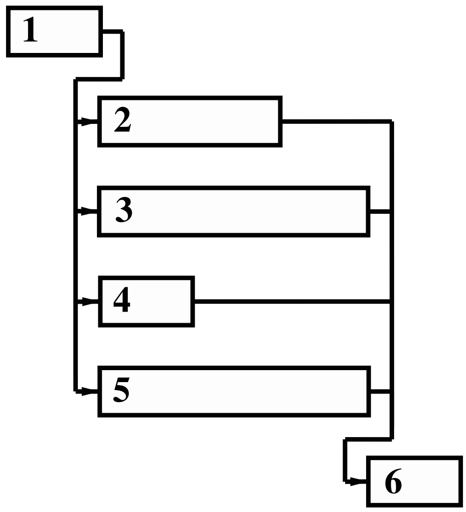
  Objetivo: documentación.

  Duración: 2 meses.

Esfuerzo: ingeniero superior, 0.5 h-m; administrativo 0.5 h-m.

Resultado: documentación de las actividades realizadas y redacción del presente documento.

La figura muestra un diagrama de Gant muestra las relaciones de dependencia entre las distintas tareas. El plazo de ejecución del proyecto es de 10 meses.



2.2 Coste de los recursos materiales

En la siguiente tabla constan los recursos materiales empleados, considerando un periodo de amortización para los equipos y licencias software de 3 años. Se muestra en primer lugar el coste total y se considera un prorrateo de 10 meses de utilización, que será el que se imputará al presupuesto.

Las características concretas de cada concepto constan en el Pliego de condiciones.

Recursos hardware	
Concepto	**Coste**
Equipo de captura × 3	4.500,00
Equipo de análisis × 3	4.050,00
Equipo de desarrollo	1.200,00
Impresora láser	510,00
Total recursos hardware	**10.260,00**

Recursos software	
Concepto	**Coste**
Sistema operativo FreeBSD 2.2.5	0,00
Sistema operativo Debian GNU/Linux 2.2.14	0,00
Herramientas de desarrollo	0,00
Sistema operativo Windows 2000	156,56
Licencía Microsoft Office	244,00
Total recursos software	**400,56**

Recursos materiales	
Concepto	**Coste**
Recursos hardware	10.260,00
Recursos software	400,56
Total recursos	10.660,56
Prorrateo 10 meses	2.961,27
Consumibles, material fungible y de oficina	300,00
Coste recursos materiales	3.261,27

2.3 Costes de mano de obra

Para la realización del proyecto se requieren los siguientes perfiles profesionales:

- Un Ingeniero Superior de Telecomunicación, encargado del planteamiento, desarrollo e implementación del trabajo técnico.

- Un Administrativo, encargado de la redacción, presentación y encuadernación del proyecto.

La estimación de costes se realiza en base a la siguiente información:

- Cotizaciones según el Régimen General de la Seguridad Social. El ingeniero pertenece al grupo 1 y el administrativo al grupo 7.

- Sueldo bruto anual según los baremos del Colegio Oficial de Ingenieros de Telecomunicación.

- Jornada laboral de 8 horas/dia y 21 días laborales/mes. Año de 1800 horas laborales.

Costes salariales		
Concepto	**Grupo 1**	**Grupo 7**
Base cotizable máxima anual	30.898,80	11.779,84
Contingencias comunes (23.6 %)	7.292,12	2.780,04
Desempleo, F.G.S. y Formación profesional (7%)	2.162,92	824,59
Accidentes de trabajo y enfermedad profesional (1%)	308,99	117,80
Coste de la Seguridad Social	9.764,03	3.722,43
Salario bruto anual	57.600,00	11.779,84
Coste salarial anual	67.364,03	15.502,27
Coste salarial por hora	37,42	8,61
Número de horas	1260	168
Coste total	**47.149,20**	**1.446,48**

Finalmente, el coste de la mano de obra es el siguiente:

Mano de obra	
Concepto	**Coste**
Ingeniero de telecomunicación	47.149,20
Administrativo	1.446,48
Coste mano de obra	**48.595,68**

2.4 Coste total de los recursos

Presupuesto de ejecución material	
Concepto	**Coste**
Coste recursos materiales	3.261,27
Coste mano de obra	48.595,68
Presupuesto de ejecución material (P.E.M.)	**51.856,95**

3 Gastos generales y Beneficio industrial

Bajo Gastos generales se incluyen todos aquellos gastos derivados de la utilización de instalaciones, cargas fiduciarias, amortizaciones, gastos fiscales, etc. El Presupuesto de Ejecución por contrata queda como sigue.

P.E.M	51.856,95
Gastos generales (16% del P.E.M.)	8.297,11
Beneficio industrial (6% del P.E.M.)	3.111,42
Presupuesto de ejecución por contrata	**63.265,48**

El presupuesto de ejecución por contrata consta de la partida anterior sumada al Presupuesto de ejecución material.

4 Honorarios por redacción y dirección del proyecto

Los Honorarios que recomienda aplicar el Colegio Oficial de Ingenieros de Telecomunicación tanto para la redacción como para la dirección de un proyecto son un 5.6% sobre el P.E.M.

5 Presupuesto total

Finalmente, sumando todas las partidas anteriores y aplicando el 16% de IVA se obtiene el presupuesto total.

Presupuesto de ejecución por contrata	63.265,48
Honorarios por dirección	2.903,99
Honorarios por redacción	2.903,99
Subtotal	69.073,46
I.V.A. (16%)	11.051,75
Presupuesto total	**80.125,21**

El presupuesto total del proyecto asciende a OCHENTA MIL CIENTO VEINTICINCO euros CON VEINTIÚN céntimos.

Madrid, julio de 2002
El ingeniero proyectista,

Fdo: Fermín Galán Márquez