UNIVERSIDAD COMPLUTENSE DE MADRID
FACULTAD DE INFORMÁTICA

Ingeniería Informática

Proyecto de Sistemas Informáticos
Curso 2010 – 2011

ANÁLISIS Y CONFORMACIÓN DE TRÁFICO EN INTERNET

Autores

Sara Manchado Illán
Manuel Marugán Cruz

Profesor Director

Juan Carlos Fabero Jiménez

Agradecimientos

En primer lugar queremos agradecer a Juan Carlos Fabero la oportunidad que nos ha brindado para poder realizar este proyecto. En su calidad de director de proyecto nos ha guiado con su experiencia en todo momento, y nos ha facilitado todo tipo de información y ayuda durante su desarrollo. Por todo esto y mucho más, gracias Juan Carlos.

Por último agradecer el apoyo de nuestros familiares y amigos por haber estado a nuestro lado siempre que lo hemos necesitado.

Autorización de difusión

Los autores del proyecto Sara Manchado Illán y Manuel Marugán Cruz autorizan a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

Firmado:

Sara Manchado Illán

Manuel Marugán Cruz

Índice general

Índice de figuras

RESUMEN

En los últimos años, Internet ha evolucionado vertiginosamente tanto a nivel de infraestructuras como a nivel de usuarios conectados, lo cual ha hecho que se convierta en la Red telemática más extendida y utilizada a nivel mundial.

Este crecimiento ha dado lugar al desarrollo de un amplio conjunto de servicios, aplicaciones (redes sociales o el correo electrónico) y nuevos paradigmas, como las comunicaciones *peer-to-peer* o VoIP, que no existían hace algunos años. También ha dado lugar al surgimiento de amenazas que se manifiestan en forma de ataques a la seguridad de nuestra Red. Estas amenazas y la continua evolución del tráfico que circula por Internet hace necesario disponer de herramientas adecuadas y flexibles que permitan conocer las características de dicho tráfico.

El marco de nuestro proyecto se centra principalmente en la creación de una herramienta que alerta de posibles ataques o comportamientos anómalos de la Red haciendo uso de un *firewall*, *Iptables*.

Palabras clave: Comportamientos de red, *Iptables*, análisis, Internet, tráfico de la Red, alertas.

ABSTRACT

These last years, Internet has tremendously evolved both at an infrastructure level and at the number of users. It makes of Internet the most extended and used telematic net in the world.

This growth has led to the development of a large set of services, applications such as social networks or electronic mail and new paradigms, such as peer to peer communications all of them nonexistent a few years ago. It has also led to the appearance of threats to the security of the net.

Those threats and the continuous evolution of the traffic that flows through the net makes necessary the creation of a tool alerting of the possible attacks or anomalous behaviors concerning the net by taking advantage of the firewall *Iptables*.

Keywords: Behavior network traffic, *Iptables*, analyzer, Internet, Net traffic, alerts.

ACERCA DE ESTE DOCUMENTO

El presente documento describe el trabajo realizado durante el proyecto de Sistemas Informáticos por Manuel Marugán Cruz y Sara Manchado Illán, bajo la dirección de Juan Carlos Fabero Jiménez en la Universidad Complutense de Madrid.

El documento sigue una estructura sencilla basada en el proceso seguido a lo largo del proyecto.

En primer lugar, se describe el proyecto desde un punto de vista global y se introducen conceptos básicos de la temática de éste para continuar entrando en detalle según avanza el documento.

Posteriormente, se describe el entorno de desarrollo mediante el cual se ha realizado el proyecto, con el fin de contextualizar mejor el ámbito central del desarrollo del trabajo. Más adelante, se expone en detalle la investigación realizada sobre determinados comportamientos de tráfico en la Red, así como las distintas herramientas utilizadas para su realización.

A continuación, se aborda el grueso de la parte técnica donde se describe el proceso de análisis, diseño e implementación de la aplicación que satisface los objetivos establecidos.

Por último se proporciona una sección de conclusiones, discusión de las principales aportaciones del proyecto así como una reflexión y bibliografía.

CAPÍTULO 1.

1. INTRODUCCIÓN

El proyecto "Análisis y conformación de tráfico en Internet" busca fomentar la seguridad en la Red y analizar algunos sistemas de comunicación en Internet. En la actualidad, Internet se ha convertido en un elemento primordial para cubrir gran parte de nuestras necesidades. Su principal uso es como fuente de información, pero también se emplea como medio de comunicación mediante redes sociales, correo electrónico o incluso llamadas telefónicas. Internet ofrece una amplia gama de servicios para sus consumidores. Sin embargo, los suministradores de Internet en ocasiones no desean que los consumidores usen Internet para determinados servicios, como es el caso de las llamadas telefónicas gratuitas mediante voz sobre IP que perjudican a las operadoras de telefonía móvil.

Por otro lado, a diario se producen miles de ataques en Internet. La inmensa mayoría son ataques indiscriminados, buscando máquinas desde las que distribuir correo basura. Buscan también utilizar dichas máquinas como plataformas para lanzar nuevos ataques y dificultar cualquier seguimiento. Debido a que estos ataques son muy difíciles de predecir, es imprescindible contar con unas mínimas medidas de seguridad.

Del análisis anterior surge la principal motivación del proyecto: proporcionar una herramienta que detecte y alerte de los posibles ataques o comportamientos no deseados que se están produciendo en la Red. De esta manera el proyecto consta de las siguientes líneas de actuación:

- Analizar el tráfico de la Red y obtener patrones de determinados comportamientos del tráfico (exploración de puertos, DoS, VoIP).
- Analizar las diferentes características y funcionalidades que poseen los IDS, centrándonos en *Snort*.
- Analizar las diferentes características y funcionalidades que poseen los *firewalls*, centrándonos en *Iptables*.
- Crear una herramienta con capacidades y funcionalidades básicas de un IDS. Además dicha herramienta debe ser capaz de generar e insertar reglas en el *firewall*. Para realizar la tarea de filtrado de paquetes se hará uso de un *firewall*, *Iptables*, de tal forma que las reglas y comandos tendrán la sintaxis necesaria para poder trabajar con dicho *firewall*.

CAPÍTULO 2.

2. PLANTEAMIENTO DEL PROYECTO

Este capítulo introduce brevemente el motivo por el que resulta interesante realizar un proyecto de estas características. Se aborda la motivación del proyecto así como los objetivos principales que se fijaron inicialmente y la metodología y planificación que se llevó a cabo en su realización.

2.1 MOTIVACIÓN DEL PROYECTO

En estos últimos años hemos vivido muchos cambios en cuanto a comunicaciones, tecnologías e Internet. Todo ello ha contribuido a que nuestras redes sean más susceptibles de ser atacadas y a encontrarnos con nuevos atacantes en potencia.

La variedad de ataques posibles contra un sistema es muy amplia, debido a que los atacantes aprovechan las debilidades de las víctimas. Por ello, se hace imprescindible diseñar un sistema de seguridad para cerrar las posibles vías de ataque.

Por otro lado, resulta interesante poder saber el uso que se está haciendo del servicio de Internet. Este punto va destinado principalmente a los suministradores de Internet, como pueden ser las operadoras telefónicas. Recordemos que la mayoría de los ingresos de las operadoras telefónicas proceden de las facturaciones de llamadas de voz y, últimamente, del consumo de datos. Por ello, podría resultarles interesante poder incorporar dicha aplicación en los encaminadores domésticos de tal forma que fueran capaces de detectar si un cliente está haciendo uso de un determinado servicio, como puede ser Telefonía sobre IP. De la misma forma que la aplicación detecta el uso de VoIP, se podrían detectar otro tipo de comportamientos interesantes.

Para poder detectar y alertar de determinados comportamientos o accesos no autorizados a un computador o a una red, existen muchas aplicaciones. Dichas aplicaciones se las conoce como “sistemas de detección de intrusos” (IDS). El funcionamiento de estas herramientas se basa en analizar el tráfico de la Red, el cual es comparado con reglas definidas previamente o patrones de comportamiento de la Red, como puede ser la exploración de puertos, DoS o VoIP. El IDS es capaz de alertar y bloquear de dichos comportamientos para convertirse en una herramienta muy poderosa.

Una de las partes de las que se compone el proyecto es el análisis de los distintos paquetes que se generan en el tráfico de red durante determinados servicios o ataques y la obtención de patrones asociados. Con ello, se procederá a crear una aplicación capaz de detectar y alertar de dichos patrones al usuario en tiempo real.

2.2 OBJETIVOS Y EVOLUCIÓN DEL PROYECTO

El objetivo principal de este proyecto era diseñar e implementar una herramienta de código libre que fuese capaz de detectar determinados comportamientos de la Red, lanzando alertas al usuario.

Sin embargo, antes de empezar a diseñar e implementar el prototipo, necesitábamos familiarizarnos con el tráfico de la Red para saber en todo momento lo que estaba sucediendo. Para ello, tuvimos que crear un entorno de trabajo en nuestros ordenadores, a través de la instalación de máquinas virtuales, para poder llevar a cabo la generación y análisis del tráfico en la Red.

Nuestro primer objetivo fue la adquisición de los conocimientos necesarios para ser capaces de entender lo que sucede en la Red y detectar comportamiento sospechoso o anómalo.

El siguiente objetivo fue obtener los patrones para los comportamientos de tráfico en la red estudiados. El orden en el que fueron estudiados los patrones estuvo relacionado con su nivel de complejidad. Empezando por el más sencillo (exploración de puertos) y acabando con el más complejo o difícil de obtener (VoIP).

Tras familiarizarnos con el flujo de red y los diversos comportamientos del tráfico, antes de ponernos a crear un prototipo de la herramienta, nos dispusimos a buscar, en el mercado, a ver si existía una herramienta con esas capacidades. El resultado de la búsqueda fue *Snort*.

Snort es un sistema de detección de intrusos (IDS). Los IDS analizan paquetes que pasan por la Red para detectar y alertar de los comportamientos "sospechosos". *Snort* era justo lo que nosotros buscábamos. Así pues, nuestra siguiente meta fue estudiar *Snort* y comprobar si realmente nos proporcionaba lo que nosotros buscábamos.

Sin embargo, durante el análisis de *Snort*, nos dimos cuenta que era una herramienta "pesada" para lo que nosotros buscábamos, ya que poseía muchas funcionalidades de las que no íbamos a hacer uso. Nosotros necesitábamos una herramienta "ligera" que no consumiese muchos recursos para poder instalarla en

dispositivos de características modestas, como podía ser un encaminador doméstico. Por ello, descartamos *Snort* y nos dispusimos a crear nuestro propio IDS pero haciendo uso de una herramienta: *Iptables*. *Iptables* sería la encargada de filtrar los paquetes según las reglas del *firewall* y registrarlas en un *Log*. Es la aplicación la responsable de la gestión de las reglas de *Iptables* y de alertar de reglas activas del *Log*.

Antes de empezar a diseñar nuestro IDS estudiamos la herramienta *Iptables*.

Una vez acabado el proceso de documentación y definición del proyecto, se comenzó con el diseño e implementación del prototipo IDS. Se decidió usar *Java* como lenguaje de programación debido a que está orientado a objetos y es en el que los integrantes del proyecto tienen más experiencia.

El siguiente objetivo fue probar nuestra propuesta con tráfico en tiempo real. Para ello se generó tráfico que se usó para poner a prueba el sistema y comprobar su rendimiento a la hora de detectar diferentes comportamientos de tráfico.

Finalmente, llegamos a la conclusión de que el objetivo principal, marcado en un principio, se había cumplido y habíamos desarrollado una aplicación capaz de generar reglas para *Iptables*, insertarlas en el *firewall*, analizar el tráfico y alertar al usuario en tiempo real de reglas activas.

2.3 METODOLOGÍA Y PLANIFICACIÓN

Para realizar el proyecto se ha decidido planificar el desarrollo del proyecto haciendo uso de la metodología de desarrollo cascada. Es un método ágil de desarrollo que divide el ciclo de vida del *software* en etapas claramente diferenciadas. Esto implica que no se puede empezar una nueva etapa hasta que no se haya finalizado la anterior. El esquema del ciclo de vida de esta metodología se representa en la siguiente figura:

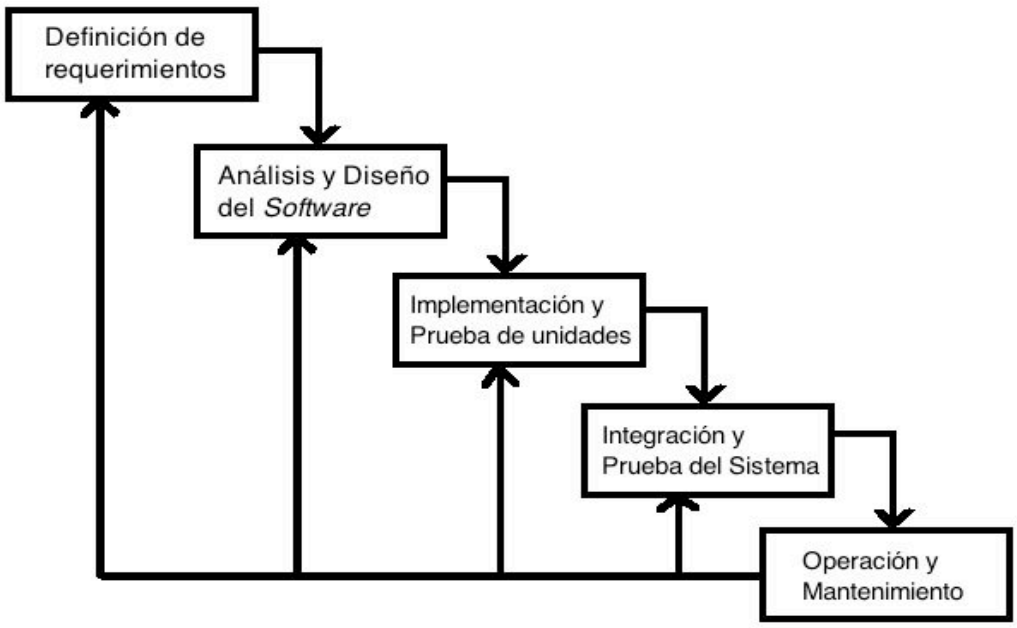


Figura 1: Ciclo de vida

El proceso de gestión y desarrollo del *software* definido en el modelo en cascada ha sido adaptado de manera natural a las necesidades y posibilidades de los integrantes del grupo del proyecto en compatibilidad con sus estudios.

En este caso la planificación se define de la siguiente manera:

- Trabajo diario y reuniones del grupo del proyecto tres veces por semana.
- Reuniones semanales formadas por los integrantes del grupo del proyecto y el director del proyecto para re planificar objetivos semanales y revisar los hitos y entregas oficiales.
- Desarrollo incremental del código y entrega de prototipos funcionales.

Para apoyar el proceso de trabajo se han usado herramientas colaborativas que han ayudado a los integrantes del grupo a mantener una visión global del estado del proyecto en todo momento.

Entre las herramientas se encuentran:

- Comunicación y planificación: correo electrónico.
- Documentación: *Google Docs* y *Microsoft Word*.
- Almacenamiento de recursos: *DropBox*.

CAPÍTULO 3.

3. ENTORNO DE DESARROLLO

3.1 INTRODUCCIÓN

En este capítulo definimos los elementos utilizados para construir el escenario necesario para poder llevar a cabo el desarrollo del proyecto. La finalidad de este punto es facilitar al lector una mejor compresión de las características de nuestro entorno de trabajo durante la realización del proyecto.

3.2 MÁQUINAS VIRTUALES

Para poder llevar a cabo el desarrollo del proyecto era necesario disponer de una infraestructura compuesta por varias máquinas físicas. Por ello, se hizo uso de las máquinas virtuales. Dicho entorno nos proporcionó el aislamiento necesario del "ruido" exterior para una realización satisfactoria de las pruebas. Sin este aislamiento cualquier intento de diferenciar las tramas correspondientes a nuestros experimentos de las tramas que circulan por la Red se convertiría en una tarea ardua y muy ineficiente.

Una máquina virtual es un *software* cuya finalidad es simular el comportamiento de un ordenador y por tanto tiene la capacidad de ejecutar programas al igual que un computador real.

Otra manera de acercarnos a la comprensión de una máquina virtual es entendiéndola como una capa de abstracción que separa el funcionamiento de un ordenador de su *hardware*.

Las máquinas virtuales se construyeron con la finalidad de simplificar el proceso de control del *hardware* de un ordenador ya que extienden y enmascaran la funcionalidad del *hardware* a través de procedimientos y datos abstractos.

El *software* más extendido para crear máquinas virtuales son:

- *VMware*
- *Virtual PC*
- *Sandbox*

Respecto a la elección del tipo de máquinas virtuales optamos por la opción de *User Mode Linux*, ya que nuestro principal requerimiento es que fuese "ligera" y segura. Dicho sistema de virtualización se detallará posteriormente.

Cabe destacar, también, que nuestros equipos requerían una primera fase de virtualización, ya que se trata de equipos *Mac* y resulta muy difícil ejecutar cualquier distribución de *Linux* en nativo debido a la naturaleza cerrada de la marca y a su política de no distribuir públicamente los drivers de los componentes. Por esta razón, en primer lugar decidimos virtualizar el sistema operativo *Ubuntu* en nuestras máquinas con la ayuda del software *VMware.*

User Mode Linux

Para todas las simulaciones del proyecto se ha hecho uso de *User Mode Linux* también conocido como UML.

User Mode Linux, instalado sobre el sistema operativo *Linux*, es un método seguro y eficiente para ejecutar procesos y versiones de *Linux*. UML permite al usuario ejecutar *software* que puede contener errores sin arriesgar la instalación principal de *Linux*. Esta es una de las razones fundamentales por las que UML es uno de los sistemas de virtualización más extendidos.

Por otra parte, *User Mode Linux* proporciona una máquina virtual que puede tener más recursos *software* y *hardware* virtuales que la máquina física. El almacenamiento en disco para la máquina está contenido completamente en un único fichero en la máquina física. Se puede asignar a la máquina virtual el acceso al *hardware* que solamente se desee. Con los permisos correctamente configurados, no se puede hacer nada en la máquina virtual que pueda dañar a la máquina física real o su *software*.

A continuación, listamos algunas de las aplicaciones para las que UML es utilizado:

- Alojamiento de servidores virtuales.
- Núcleo de desarrollo.
- Experimentos con nuevos *kernels* y distribuciones.
- Educación.

Al iniciar UML, se creará un fichero cuyo tamaño es el mismo que el del tamaño del disco de UML. Un problema que puede surgir al utilizar varias máquinas virtuales, que utilizan la misma imagen de disco, es la duplicación innecesaria de datos en la memoria de la máquina física. Dicho problema es resuelto gracias al uso de la tecnología COW (*Copy On Write*).

La tecnología COW emplea ficheros dispersos (*Sparse)*. Dichos ficheros están distribuidos en bloques donde existe un índice que indica qué bloques están realmente ocupados. Por ello, podemos crear ficheros de un determinado tamaño a pesar de que el tamaño real que van a ocupar en el disco equivaldrá simplemente al tamaño que ocupen los bloques que contengan información válida. De este modo, si se crea un fichero vacío de 1GB sólo ocupará 4KB.

Arrancar estas máquinas virtuales desde la misma imagen con ficheros COW reducirá el número de copias en la página de caché del host a una, pero seguirá habiendo n copias en la página de caché de UML, una por cada UML.

A continuación, haremos una breve descripción de tres ficheros con los que trabaja UML.

- Root_fs: fichero que contiene las características comunes de todas las máquinas UML.
- Fichero COW: contiene las características propias de cada máquina UML.
- Fichero SWAP: es el fichero de intercambio ubicado en cada una de las máquinas UML.

Queremos señalar que, tal vez, debido a nuestra inexperiencia con dicho sistema de virtualización, hemos encontrado ciertos problemas como el deterioro del sistema de ficheros y por tanto la inutilización de las máquinas virtuales. A pesar de ello, seguimos considerando dicho sistema de virtualización el adecuado en el marco de un proyecto basado en las redes.

A continuación, podemos ver en la figura el entorno de trabajo descrito.

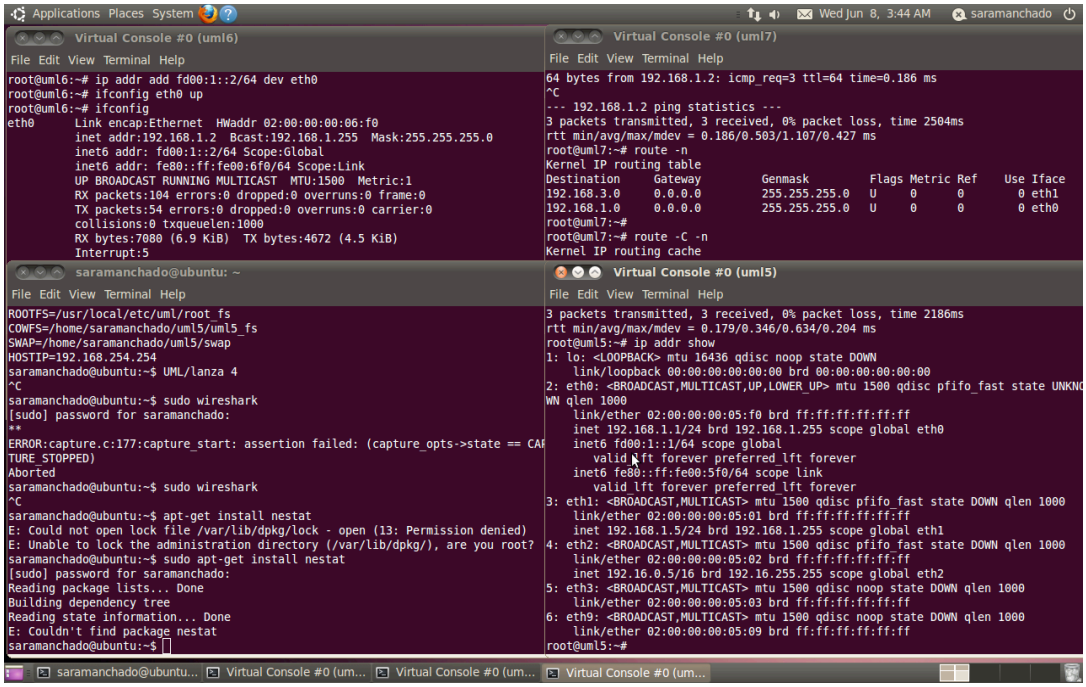


Figura 2: Entorno de trabajo

3.3 PROTOCOLOS

Conocer el funcionamiento y gestión de los siguientes protocolos se revela fundamental para la comprensión del proyecto. A continuación, exponemos una breve y concisa información acerca de cada uno de ellos con el fin de informar al lector inexperto en estas materias.

3.3.1 IP

El protocolo de Internet es un protocolo no orientado a conexión y que funciona a través de una red conmutada de paquetes. Es, por tanto, un protocolo de máximo esfuerzo de entrega de paquetes no confiable. Es uno de los protocolos de Internet más importantes ya que permite el transporte de paquetes de datos a pesar de que se haga sin garantías.

Los datos circulan en forma de datagramas. Los datagramas contienen dos partes: la cabecera, que contendrá información relativa a su transporte con las direcciones IP origen y destino y en segundo lugar los datos que son relevantes para los protocolos de las capas superiores. Los encaminadores, encargados de analizar y, ocasionalmente, modificar los datos contenidos en los datagramas, aportan la posibilidad de que estos puedan transitar. Estos se ayudan de las direcciones IP origen y destino para encaminar correcta y eficientemente los paquetes de datos.

Cabe destacar a su vez, el hecho de que IP no posee ningún mecanismo para determinar si un paquete alcanza o no su destino y únicamente proporciona seguridad de sus cabeceras y no de los datos transmitidos. De hecho, estos podrían alcanzar su destino dañados o incluso no alcanzar su destino.

En caso de requerir fiabilidad, ésta es proporcionada por los protocolos de la capa de transporte, como es el caso de TCP.

A continuación se muestra el datagrama IP con sus correspondientes campos:

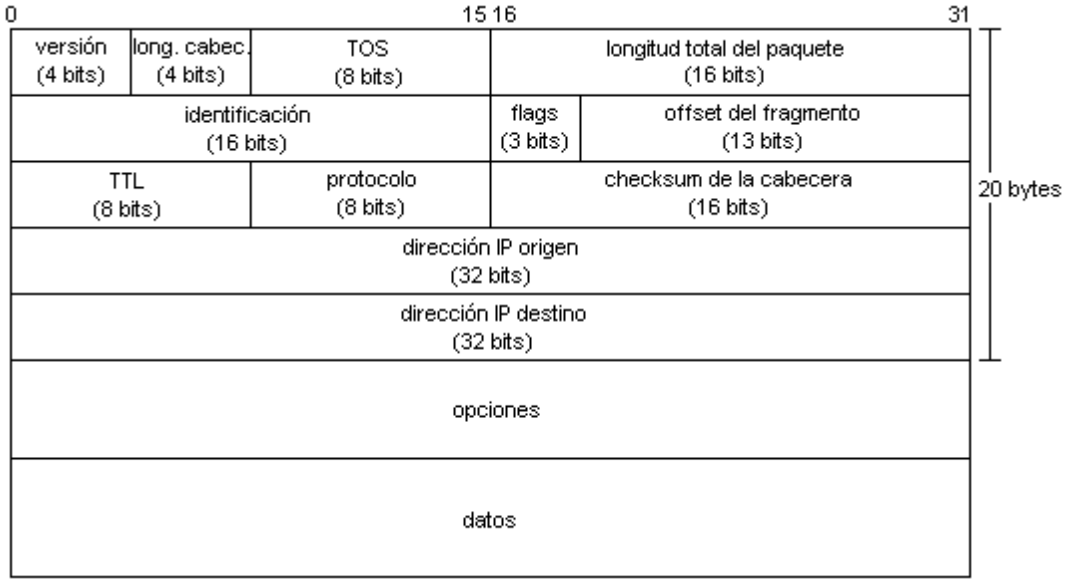


Figura 3: Datagrama IP

3.3.2 TCP

TCP (*Transimission Control Protocol*) es un protocolo de transporte orientado a conexión y fiable. Se trata de uno de los protocolos más importantes de Internet. Dicho protocolo garantiza que los datos serán entregados a su destino en el mismo orden en que se transmitieron y libres de errores. Este protocolo se encuentra en la capa de Transporte, o nivel 4, con respecto al modelo de referencia OSI. La unidad de transferencia en este protocolo es el segmento.

Dado que el protocolo TCP garantiza un servicio extremo a extremo fiable, éste debe realizar las siguientes funciones:

- Detectar y retransmitir segmentos de datos perdidos o erróneos.
- Detectar y descartar segmentos duplicados.
- Entregar los segmentos de forma ordenada a la capa de aplicación.

El formato de los segmentos TCP se muestra en el siguiente esquema:

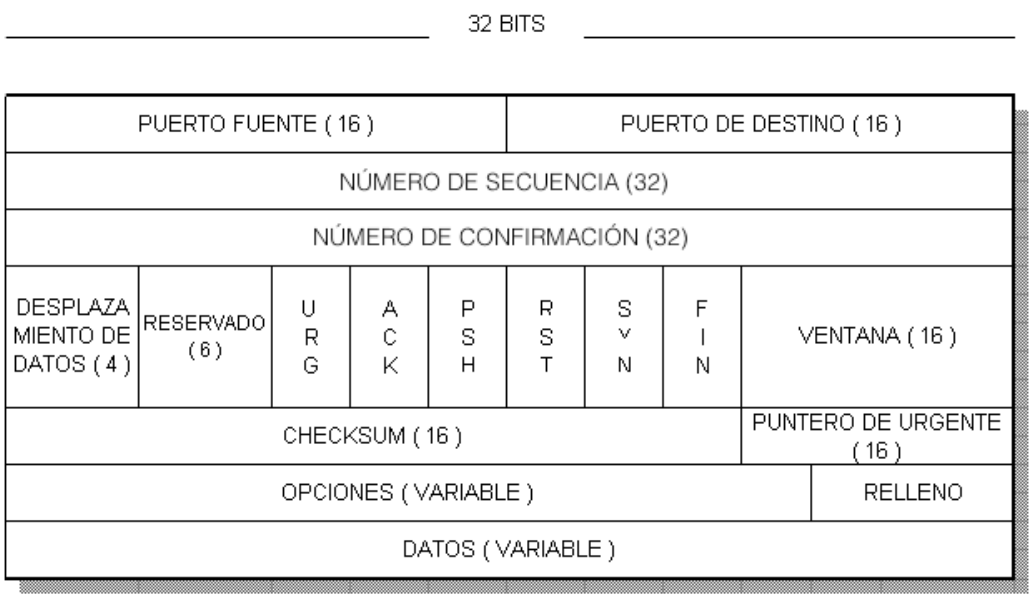


Figura 4: Formato segmentos TCP

Como podemos apreciar en la figura, el segmento TCP está compuesto por varios campos, de los cuales nos centraremos en el campo *flags* del encabezado (*header*) debido a que es necesario conocer el significado de cada uno de los *flags* para poder analizar el tráfico de red y generar reglas correctamente.

El campo *flags* está formado por seis banderas de un bit. Cada una de estas banderas puede obtener el valor 0 ó 1, dependiendo de si el bit está o no activado.

- **URG**: define un bloque de datos como urgente.
- **ACK**: confirmar una acción.
- **PSH**: fuerza el envío inmediato de datos tan pronto como sea posible.
- **RST**: reiniciar una conexión.
- **SYN**: iniciar una conexión TCP.
- **FIN**: finalizar una conexión TCP.

3.3.3 UDP

El protocolo UDP es un protocolo no orientado a conexión de la capa de Transporte del modelo de referencia OSI. Se trata de un protocolo muy simple ya que no proporciona detección de errores, es decir, no garantiza un servicio extremo a extremo fiable. La unidad de transferencia en este protocolo es el datagrama.

El formato del datagrama UDP es el siguiente:

Puerto fuente (16 bits)	Puerto destino (16 bits)
Longitud (16 bits)	Suma de verificación (16 bits)
Datos	

Figura 5: Formato datagrama UDP

Dicho protocolo es aplicado, a diferencia del protocolo TCP, en aplicaciones que se basan en la rapidez a la hora de la entrega y no tanto en la fiabilidad. Algunos ejemplos de dichas aplicaciones son: DNS, SNMP, RIP, RTP, etc.

3.3.4 ICMP

El protocolo ICMP es el encargado del control de errores durante la comunicación entre dos extremos, es decir, informa al origen si se ha producido algún error durante la entrega del mensaje. Se debe destacar que el protocolo ICMP sólo informa acerca de los errores sucedidos pero nunca los solventará. Dicha tarea es relegada a las capas superiores.

Cabecera ICMP:

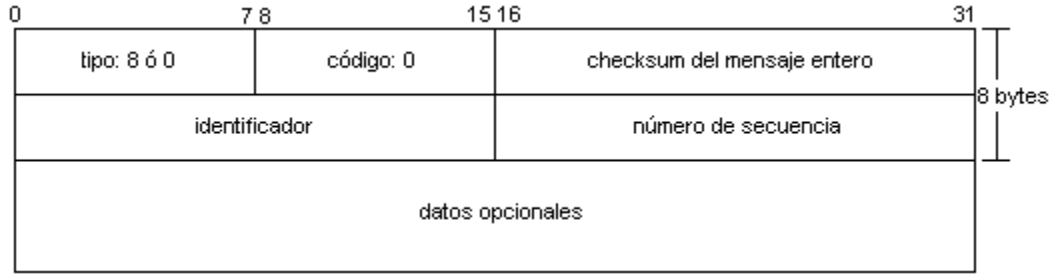


Figura 6: Cabecera ICMP

3.3.5 SIP

SIP, también conocido como Protocolo de Inicio de Sesiones, es un protocolo de control y señalización usado fundamentalmente en los sistemas de Telefonía IP. Dicho protocolo permite crear, modificar y finalizar sesiones multimedia con uno o más participantes y podemos contar entre sus principales ventajas la simplicidad y consistencia.

En concreto en el propio RFC 3261 se define como:

"*SIP es un protocolo de control de la capa de aplicación que puede establecer, modificar, y terminar sesiones multimedia como llamadas telefónicas por Internet. SIP puede también invitar participantes a sesiones existentes, como conferencias.*"

El protocolo SIP define principalmente seis métodos, los cuales están definidos en el RFC:

- **INVITE**: establece una sesión.
- **ACK**: confirma una solicitud INVITE.
- **BYE**: finaliza una sesión.
- **CANCEL**: cancela el establecimiento de una sesión.
- **REGISTER**: comunica la localización de usuario.
- **OPTIONS**: comunica la información acerca de las capacidades de envío y recepción de teléfonos SIP.

A su vez, el protocolo SIP define seis códigos de respuestas:

- **1xx**: respuestas informativas.
- **2xx**: respuestas de éxito.
- **3xx**: respuestas de redirección.
- **4xx**: errores de solicitud.
- **5xx**: errores de servidor.
- **6xx**: errores globales.

Basándonos en lo anterior, cabe destacar que SIP no es un protocolo de propósito general y su objetivo es tratar de ayudar en el establecimiento y finalización de una comunicación.

A continuación, mostramos un ejemplo de una llamada SIP en la que se puede apreciar la evolución de ésta y con el fin de proporcionar un mayor entendimiento de dicho protocolo:

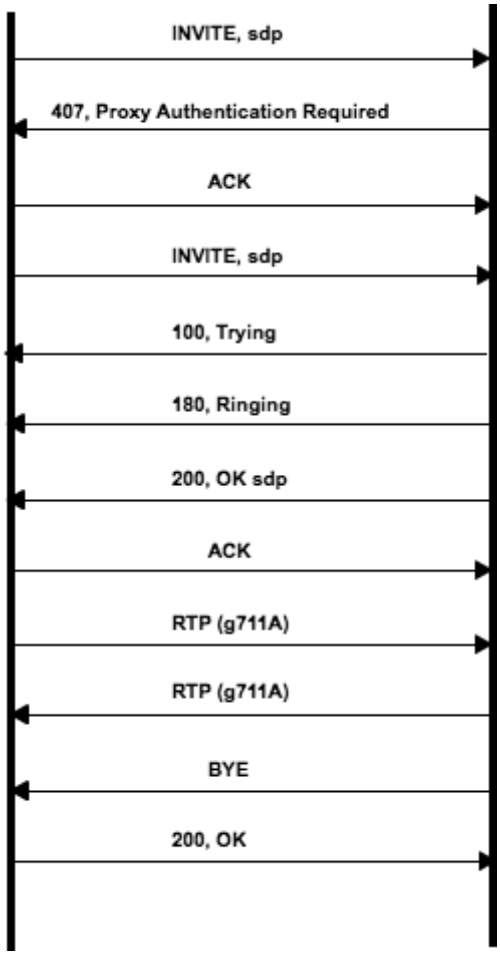


Figura 7: Llamada SIP

En la figura anterior podemos apreciar cómo en primer lugar el teléfono llamante envía un INVITE con el fin de establecer la conexión.

A continuación, el otro extremo (llamado), solicita la autenticación mediante la respuesta 407 (Autenticación Proxy Requerido).

El llamante responde con un ACK y el llamado envía una respuesta informativa 100 a posteriori. Cuando el teléfono llamado comienza a sonar manda una respuesta 180 (teléfono sonando).

En el momento en que el receptor levanta el teléfono, el llamado informa de ello con una respuesta 200. A continuación, el llamante confirma con un ACK.

Tras esto la comunicación se trasmite bajo el protocolo RTP.

Cuando el llamado cuelga, se manda una solicitud de finalización de sesión BYE. Dicha solicitud es confirmada con un OK y se finaliza la comunicación.

SIP es, por tanto, un protocolo que define mecanismos para garantizar la fiabilidad de las comunicaciones y depende del protocolo SDP (*Sesion Description Protocol*).

SDP es el responsable de describir el contenido multimedia de la sesión, por ejemplo, que puertos, códecs o direcciones IP se utilizarán durante la conexión. Un dato interesante para el análisis del tráfico de red, es que las aplicaciones SIP utilizan el puerto 5060 tanto para TCP como para UDP.

CAPÍTULO 4.

4. TIPOS DE COMPORTAMIENTOS DE RED

4.1 INTRODUCCIÓN

El objetivo primordial de este capítulo es dar al lector una breve introducción teórica acerca de unos comportamientos de red específicos. Dichos comportamientos de red han sido analizados por los miembros del grupo con el fin de poder detectarlos por medio de nuestra aplicación.

De este modo, este capítulo trata de aportar bases teóricas acerca de los comportamientos de tráfico de red: exploración de puertos, denegación de servicio y Telefonía sobre IP.

Como paso previo a clasificar o caracterizar el tráfico en la Red, resulta fundamental dar una somera definición del concepto de tráfico de red.

El tráfico de red hace referencia a los paquetes que circulan por la red durante un determinado período de tiempo. Durante el análisis del tráfico de red se pueden encontrar paquetes maliciosos o anómalos dentro del tráfico normal. Estos paquetes tendrán que ser analizados con más detalle para ver si se tratan de paquetes espontáneos o paquetes con un objetivo oculto.

En caso de que dichos paquetes representen una amenaza, podemos concluir, que forman parte de un ataque.

Un ataque se define como la acción de aprovecharse de una vulnerabilidad de un sistema informático con fines desconocidos por el operador del sistema. Por lo general, un ataque se realiza con el propósito de causar un daño. Los ataques, en su mayoría se lanzan automáticamente desde equipos infectados sin que el propietario sepa lo que está ocurriendo. En casos típicos, son ejecutados por piratas informáticos. Los ataques pueden producirse por diversos motivos, entre los cuales:

- Acceder al sistema.
- Manipulación de información.
- Obtener información confidencial de la víctima.
- Degradar el funcionamiento normal de un servicio.
- Utilizar recursos del sistema.

Tal y como hemos especificado en el apartado correspondiente a la motivación del proyecto, no sólo resulta interesante estudiar comportamientos maliciosos, sino, también, caracterizar comportamientos de red asociados a los servicios que ofrecen determinadas empresas a los usuarios haciendo uso de Internet.

En nuestro caso nos decantamos por analizar el comportamiento de red del servicio de Telefonía sobre IP ofrecido gracias a la tecnología VoIP.

A continuación, se muestra una información detallada de cada uno de los comportamientos de red que acabamos de mencionar.

4.2 EXPLORACIÓN DE PUERTOS

Una exploración de puertos es una actividad que permite al atacante obtener información acerca de los servicios ofrecidos por su víctima e incluso detalles de la arquitectura de su red. Esta técnica no se puede considerar en sí misma como un ataque aunque se caracteriza por ser el paso previo en multitud de ellos.

Esta técnica también puede utilizarse como técnica defensiva ya que cualquier usuario puede hacer uso de ella para detectar qué puertos están abiertos innecesariamente y cerrarlos a continuación.

Cabe destacar que durante nuestros experimentos utilizábamos la herramienta *Netstat* para visualizar el listado de puertos abiertos y el estado de los mismos de nuestras máquinas.

La exploración de puertos es una técnica que consiste en el envío de paquetes de diferentes protocolos a los puertos de la víctima con el objetivo de averiguar qué puertos están abiertos o cerrados.

Existen multitud de técnicas para realizar una exploración de puertos. A continuación se muestran las más populares:

<u>Exploración de puertos TCP</u>

- Técnica *TCP connect scan*:

  Esta técnica se basa en la exploración de puertos mediante el establecimiento de una conexión TCP. En caso de que la conexión se realice satisfactoriamente se anotará el puerto como abierto. El servicio asociado a dicho puerto puede ser calculado en función del número de

dicho puerto aunque sólo se trataría de una suposición y no de una certeza.

Una de las principales desventajas de dicha técnica es que al realizar el establecimiento de conexión el atacante queda, en cierto modo, al descubierto y su detección resulta asequible. Otra desventaja a tener en cuenta es el hecho de que en caso de existencia de cortafuegos que filtren los paquetes de intento de conexión el atacante es incapaz de averiguar si el puerto está cerrado o abierto.

- Técnica *TCP SYN scan*:

Esta es una técnica similar a la anterior pero con la salvedad de que simplemente se envían paquetes con el *bit* SYN activado de solicitud de inicio de conexión. En caso de recibir como respuesta un paquete con los *bits* RST-ACK activados se interpreta que dicho puerto está cerrado y no hay ningún servicio que escuche por dicho puerto.

En caso de que se reciba un paquete con los *bits* SYN-ACK activados se interpreta como que dicho puerto está abierto y tiene un servicio asociado. Como no queremos establecer conexión ni tampoco quedar registrados por el sistema objetivo, reiniciaremos la conexión enviando un paquete con los *bits* RST-ACK.

- Técnica *TCP FIN scan*:

Esta técnica, también conocida como FIN *stealth*, consiste en enviar un segmento con el *bit* FIN activado. En consecuencia, la víctima puede responder con un paquete de *reset* (RST) si dicho puerto está cerrado o no responder en cuyo caso el puerto puede estar abierto o filtrado por un *firewall*.

Esta técnica, a diferencia de la técnica *Connect*, no deja rastro por lo que no es tan fácil de detectar al atacante. Además, si se combina con otras técnicas es capaz de distinguir si los puertos están cerrados o filtrados.

- Técnica *Xmas scan*:

Su configuración consiste en poner a uno los *bits* PUSH, FIN y URG.

- Técnica *Null scan*:

  En esta técnica todos los indicadores de la cabecera TCP van a estar a cero. Con ello la exploración debería recibir como resultado un paquete de *reset* en los puertos no activos.

Existen más técnicas de exploración de puertos en TCP como ACK *stealth* y FIN-ACK *stealth* que también hemos estudiado pero que no detallamos en esta lista.

Exploración de puertos UDP

En virtud de que el protocolo UDP es mucho más sencillo que el protocolo TCP sólo existe un modo de realizar una exploración de puertos UDP y esta consiste en enviar datagramas UDP sin ninguna información al campo de datos. En el caso de que el puerto esté cerrado, se recibirá un mensaje de control ICMP indicando que el puerto no es alcanzable (*port unreachable*). Si por el contrario, el puerto está abierto, no se recibirá ninguna respuesta.

Debido a que UDP es un protocolo no orientado a conexión, la fiabilidad de este método depende de numerosos factores, como pueden ser: la utilización de la Red y sus recursos, la carga existente o la existencia de filtros de paquetes en cortafuegos.

Asimismo, y a diferencia de las exploraciones TCP, se trata de un proceso mucho más lento, puesto que la recepción o no de los paquetes enviados se decide mediante el vencimiento de temporizadores (*timeouts*).

En el caso en el cual el atacante detecte un elevado número de puertos UDP abiertos, éste podría llegar a la conclusión de la existencia de un *firewall* entre su máquina y la de la víctima. Para confirmar esta última posibilidad, se puede enviar un datagrama UDP al puerto cero. Con ello, se debería recibir un mensaje de respuesta ICMP de puerto no alcanzable. En el caso de no recibir dicha respuesta significa que efectivamente existe un *firewall* que analiza el tráfico.

La herramienta utilizada para realizar la exploración de puertos ha sido *Nmap*. En el apartado 5.5.3 se puede ver el uso de dicha herramienta para la realización de una exploración de puertos.

4.3 DENEGACIÓN DE SERVICIO

Un ataque de denegación de servicio (*Denial of service)* tiene como propósito atacar un sistema de computadoras o red con el objetivo de degradar de forma parcial o total los servicios prestados por ese recurso. En dichas circunstancias, el servidor de la víctima se sobrecarga, perdiendo la capacidad de procesar futuros servicios, como consecuencia de una saturación de sus puertos por flujo de información.

Existen tres tipos básicos de denegación de servicio:

- Consumo de recursos: el atacante intenta consumir los recursos hasta agotarlos, tales como, ancho de banda o memoria.
- Destrucción o alteración de la configuración: intenta modificar la información de la máquina.
- Destrucción o alteración física de los equipos: intenta denegar el servicio destruyendo físicamente el servidor o alguno de los componentes.

A continuación, exponemos algunos ejemplos de ataques de denegación de servicio:

- Inundar una red, impidiendo el tráfico legítimo de ésta.
- Interrumpir la conexión entre dos máquinas y por lo tanto, impedir el acceso a un determinado servicio.
- Impedir el acceso de un usuario a un servicio.
- Interrumpir los servicios de un sistema específico.

A continuación, exponemos algunos de los tipos de ataques de denegación de servicio más representativos:

- Técnica *IP Flooding*:

  El ataque de *IP Flooding* se basa en una inundación masiva de la Red mediante datagramas IP.

  Este ataque se suele realizar en redes locales o en conexiones con un gran ancho de banda. Se basa en la generación de tráfico basura con la finalidad de conseguir que el servicio se degrade. De dicho modo, el ancho de banda disponible se ve reducido lo que deriva en una ralentización de las comunicaciones existentes de toda la red.

  Una de las principales variantes del ataque *IP Flooding* tradicional consiste en utilizar la dirección *broadcast* de la Red como dirección de

destino de los datagramas IP. Cuando el encaminador decide enviar el paquete, la dirección destino le obliga a encaminarlo hacia todos los *hosts* de la Red lo que provoca un enorme consumo de ancho de banda y la subsecuente degradación del rendimiento del servicio.

También existen otras variantes en las que se envían peticiones ICMP de tipo *echo-request* a varios ordenadores suplantando la dirección IP de origen, sustituida por la dirección de difusión (*broadcast*) de la Red que se quiere atacar. De esta forma, todas las respuestas individuales se ven amplificadas y propagadas a todos los ordenadores conectados a la Red.

- Técnica *Smurf*:

La técnica *Smurf* se trata de una variante de *IP Flooding* y se basa en realizar una suplantación de direcciones IP en una petición ICMP. La dirección de origen tomará el valor de la dirección IP de la víctima. La dirección IP destino tomará el valor de la dirección de difusión de red que se utilizará como puente para colapsar a la víctima.

De este modo, todos los *hosts* de la Red responderán simultáneamente a la misma máquina lo que provocará un altísimo consumo de ancho de banda y la saturación de los recursos de la víctima.

- Técnica *TCP/SYN Flooding*:

El ataque de TCP/SYN *Flooding* se basa en el sistema de establecimiento de conexión TCP. Dicho sistema de establecimiento de conexión sigue el comúnmente denominado “triple apretón de manos” (*three way handshake*).

El cliente comienza enviando la petición de conexión mediante un paquete con el *bit* SYN activado como parte de la negociación en tres pasos. Si dicho puerto está activado o abierto, el servidor enviará un paquete SYN/ACK respondiendo a la petición. Por último, el cliente responde al servidor enviando un ACK dando por concluido y satisfactorio el establecimiento de conexión.

El objetivo de esta técnica es inundar la cola de servicios de la víctima enviando paquetes con el *bit* SYN activado simulando un intento de conexión. Dado que se trata de exceder los límites establecidos para el número de conexiones el atacante falseará la dirección origen y no contestará los SYN/ACK provenientes de la víctima.

- Técnica "*ping de la muerte*":

El ataque de denegación de servicio "*ping de la muerte*" (*ping of death*) es uno de los ataques más conocidos ya que afectó a la mayoría de sistemas operativos en su creación. Actualmente, los sistemas operativos son capaces de detectar este *exploit* por lo que su daño ha visto reducido.

El ataque *ping* de la muerte consiste en construir, mediante el comando *ping*, un datagrama IP superior a los 65535 *bytes*, el cual se fragmenta en trozos con el objetivo de colapsar el sistema de la víctima. La víctima trata de reconstruir el paquete original, que había sido fragmentado multitud de veces, y se genera una saturación del buffer causando la degradación total sistema.

4.4 VOZ SOBRE IP

Voz sobre Protocolo de Internet, también conocido como VoIP, es la tecnología que hace posible que la voz pueda viajar a través de Internet.

A menudo, se confunde el servicio con la tecnología ya que se identifica a VoIp como el servicio, mientras que, en realidad, se trata simplemente del sistema que posibilita la existencia de la telefonía a través de Internet. La voz se envía en forma digital en lugar de enviarla en forma analógica tal y como ocurría en la Red Telefónica Pública Conmutada.

Diferencias entre telefonía tradicional y telefonía en Internet

En la telefonía tradicional, una centralita es la encargada de establecer la conexión entre los dos interlocutores de la llamada. Los recursos utilizados para el establecimiento de dicha llamada no pueden ser utilizados de nuevo hasta que ésta finalice.

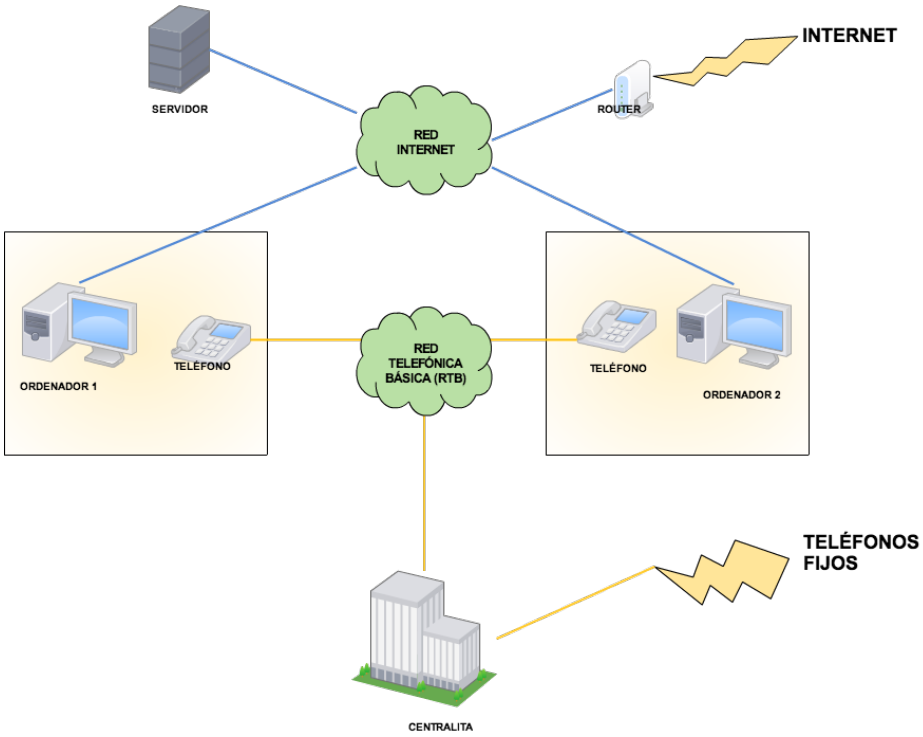


Figura 8: Telefonía tradicional

En el caso de una comunicación VoIP, su funcionamiento es el siguiente:

La voz es digitalizada y transformada en el origen de la llamada en un conjunto de paquetes de información que son transmitidos a través de la Red.

Para alcanzar el destino, cada paquete puede seguir un camino distinto, dependiendo de las condiciones de la Red, y compartiendo el medio con otros paquetes de datos.

Finalmente, en el destino estos paquetes de datos son ordenados y transformados de nuevo en voz.

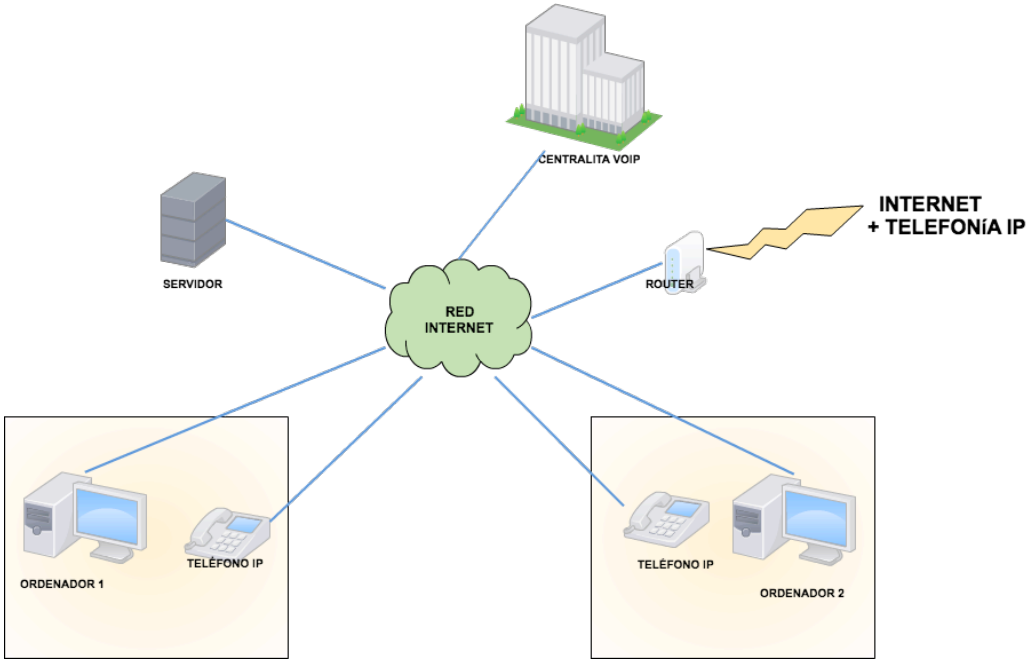


Figura 9: Telefonía a través de Internet

Ventajas de la tecnología VoIP:

La principal ventaja de la voz sobre IP reside en la reducción de costes en el importe de las llamadas realizadas. Un sistema VoIP permite establecer comunicaciones de voz con cualquier parte del mundo sin suponer el gasto extra que significa establecer una conferencia telefónica clásica con dicho lugar. En el caso de que el receptor también disponga de Telefonía IP compatible, la comunicación es gratuita. Todo ello supone una importante reducción en la factura telefónica, especialmente de aquellas empresas que necesiten frecuentemente establecer conversaciones telefónicas con el extranjero.

Actualmente se encuentran en desarrollo multitud de códecs para VoIP que permiten que la voz se codifique en paquetes de datos pequeños. Esto provoca que el ancho de banda requerido por una comunicación de voz sobre IP sea sumamente pequeño y es uno de los principales motivos del éxito de dicha tecnología.

Desventajas de la tecnología VoIP:

Una de las desventajas más importantes de dicha tecnología es que la calidad de transmisión es un poco inferior a la de la telefonía tradicional. De hecho, las conversaciones pueden verse a menudo distorsionadas o incluso cortadas. Para

establecer conversaciones VoIP satisfactorias es requisito indispensable una cierta estabilidad y calidad en la línea de datos.

En cualquier caso, con la evolución tecnológica la Telefonía IP será capaz de solventar todos sus problemas y se estima que reemplace, en la mayoría de países, a la telefonía convencional en un corto espacio de tiempo. De hecho, la mayor parte de las operadoras de telefonía ya utilizan esta tecnología en sus redes troncales.

CAPÍTULO 5.

5. HERRAMIENTAS UTILIZADAS PARA EL DESARROLLO

5.1 INTRODUCCIÓN

A continuación, se detallan las características de las herramientas que han sido necesarias para poder llevar a cabo el desarrollo del proyecto.

5.2 WIRESHARK

Wireshark junto con *Tcpdump* han sido las herramientas elegidas para realizar el análisis de tráfico de red durante el desarrollo del proyecto. Las capacidades de *Wireshark* son muy similares a las de *Tcpdump*. La principal diferencia entre ambas es la interfaz gráfica que ofrece *Wireshark* de la cual carece *Tcpdump*, lo que convierte a *Tcpdump* en una herramienta más "ligera".

Wireshark es una herramienta que permite analizar el tráfico de la Red. Captura los paquetes que están circulando por la Red, permitiéndonos averiguar qué es lo que está sucediendo en la Red.

Sus principales características son:

- Software libre.
- Disponible en múltiples sistemas operativos, entre los cuales: *Windows*, *Linux*, *Solaris* y *Mac OS*.
- Permite obtener información detallada de cada uno de los campos de los paquetes que captura.
- Captura paquetes directamente desde una interfaz de red.
- Permite obtener estadísticas.
- Sus funciones gráficas son muy poderosas, entre las cuales podemos destacar la identificación de paquetes mediante el uso de diferentes colores.

Ejemplo práctico:

En la siguiente figura se puede ver cómo gracias a la herramienta *Wireshark* podemos ver el tráfico de paquetes que están circulando en una exploración de puertos.

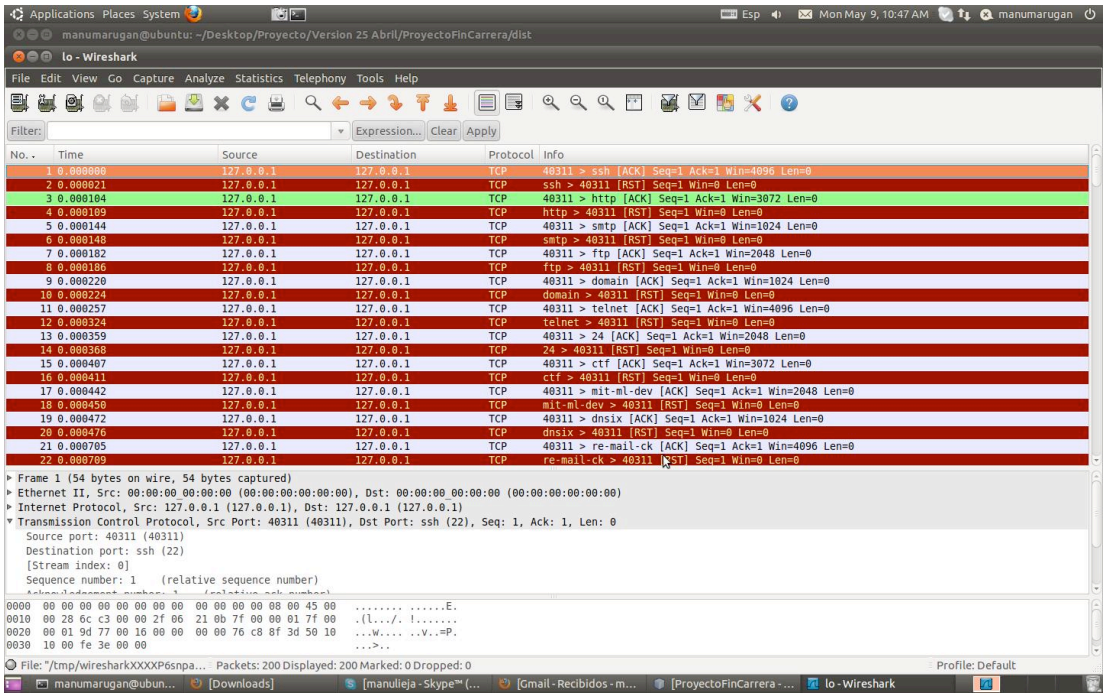


Figura 10: Tráfico de paquetes

5.3 NMAP

La herramienta *Nmap* ha sido utilizada en el proyecto como explorador de puertos, la cual nos permitía determinar, de una forma rápida y sencilla qué servidores estaban activos y qué servicios ofrecían.

Nmap es la herramienta elegida por excelencia para realizar exploración de puertos. Esto es debido a que *Nmap* implementa la gran mayoría de técnicas más comunes en una exploración de puertos.

Como tal, esta herramienta es utilizada por los expertos en seguridad para mejorar la seguridad de la Red. Sin embargo, también puede ser utilizado con malas intenciones por los *hackers* para orientar los ataques a sistemas remotos.

Sus principales características son:

- Software libre.
- Flexible. Soporta múltiples técnicas para el mapeo de redes compuestas por *firewalls*, *routers* u otros obstáculos.

- Multiplataforma. Disponible en múltiples sistemas operativos, entre los cuales: *Microsoft Windows*, *Linux*, *Solaris* y *Mac OS X*.
- Poderosa. *Nmap* es una herramienta capaz de explorar redes compuestas por miles de máquinas.

Ejemplo práctico:

A continuación se muestran exploraciones de puertos usando diferentes técnicas.

En las siguientes exploraciones de puertos se exploran los puertos del 1 al 10 de la máquina desde la que se lanza el comando.

Para realizar una exploración de puertos mediante la técnica *ACK stealth*, introducimos en la terminal:

sudo nmap –sA –p 1-10 127.0.0.1

```
manumarugan@ubuntu:~/Desktop/Proyecto/Version 25 Abril/ProyectoFinCarrera$ sudo nmap -sA -p 1-10 127.0.0.1

Starting Nmap 5.21 ( http://nmap.org ) at 2011-05-09 08:33 PDT
Nmap scan report for localhost.localdomain (127.0.0.1)
Host is up (0.000041s latency).
PORT   STATE      SERVICE
1/tcp  unfiltered tcpmux
2/tcp  unfiltered compressnet
3/tcp  unfiltered compressnet
4/tcp  unfiltered unknown
5/tcp  unfiltered unknown
6/tcp  unfiltered unknown
7/tcp  unfiltered echo
8/tcp  unfiltered unknown
9/tcp  unfiltered discard
10/tcp unfiltered unknown

Nmap done: 1 IP address (1 host up) scanned in 0.09 seconds
```

Figura 11: Exploración de puertos mediante técnica ACK stealth con NMAP

Para realizar una exploración de puertos mediante la técnica *FIN stealth*, introducimos en la terminal:

sudo nmap –sF –p 1-10 127.0.0.1

```
manumarugan@ubuntu:~/Desktop/Proyecto/Version 25 Abril/ProyectoFinCarrera$ sudo nmap -sF -p 1-10 127.0.0.1

Starting Nmap 5.21 ( http://nmap.org ) at 2011-05-09 08:34 PDT
Nmap scan report for localhost.localdomain (127.0.0.1)
Host is up (0.000039s latency).
PORT   STATE  SERVICE
1/tcp  closed tcpmux
2/tcp  closed compressnet
3/tcp  closed compressnet
4/tcp  closed unknown
5/tcp  closed unknown
6/tcp  closed unknown
7/tcp  closed echo
8/tcp  closed unknown
9/tcp  closed discard
10/tcp closed unknown

Nmap done: 1 IP address (1 host up) scanned in 0.09 seconds
```

Figura 12: Exploración de puertos mediante técnica FIN stealth con NMAP

Para realizar una exploración de puertos mediante la técnica *NULL stealth*, introducimos en la terminal:

sudo nmap –sN –p 1-10 127.0.0.1

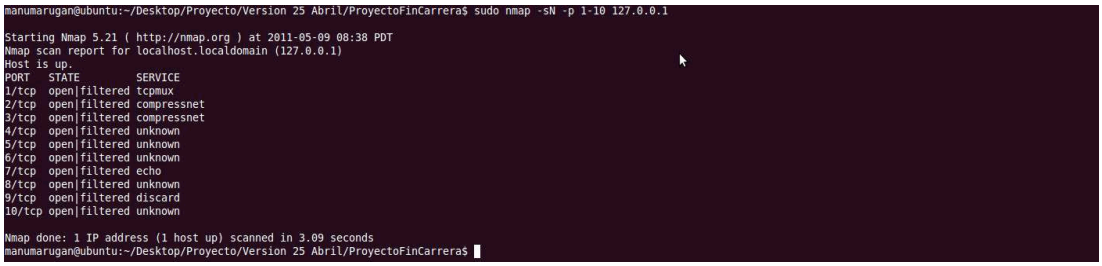


Figura 13: Exploración de puertos mediante técnica NULL stealth con NMAP

Para realizar una exploración de puertos mediante la técnica *CONNECT*, introducimos en la terminal:

sudo nmap –sT –p 1-10 127.0.0.1

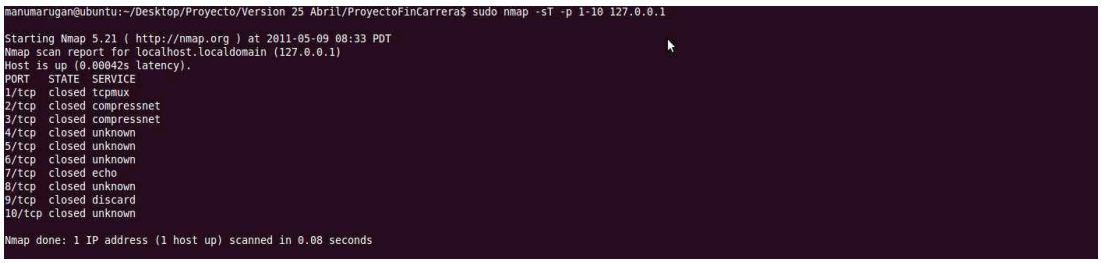


Figura 14: Exploración de puertos mediante técnica CONNECT con NMAP

5.4 TCPDUMP

Tcpdump es una herramienta utilizada para analizar el tráfico que circula por la Red. Captura los paquetes de red transmitidos en tiempo real dándole al usuario conocimiento de lo que sucede en la Red.

Para llevar a cabo la tarea de la captación de paquetes, *Tcpdump* hace uso de la librería *libpcap*, al igual que *Wireshark*.

Por otra parte, *Tcpdump* está disponible en una amplia gama de sistemas operativos, tales como, *Linux*, *Solaris*, *BSD*, *Mac OS X* entre otros.

Ejemplo práctico:

En primer lugar, hacemos una exploración de puertos mediante la técnica *ACK stealth* haciendo uso de *Nmap*.

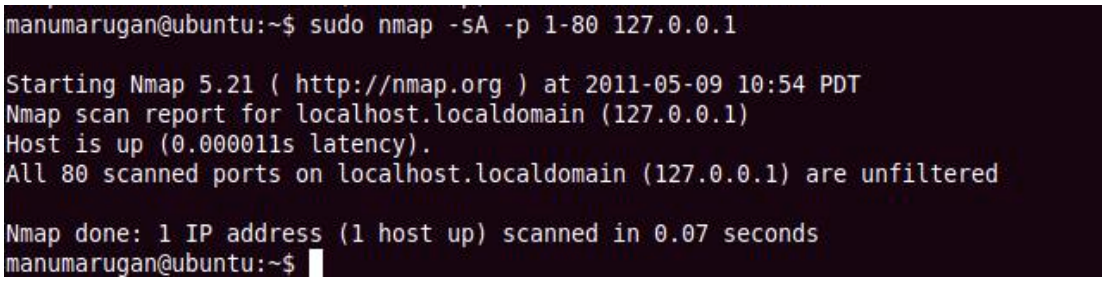


Figura 15: Exploración de puertos mediante técnica ACK stealth con NMAP

A continuación, escribimos en la terminal *sudo tcpdump –i lo* para observar el tráfico de la Red.

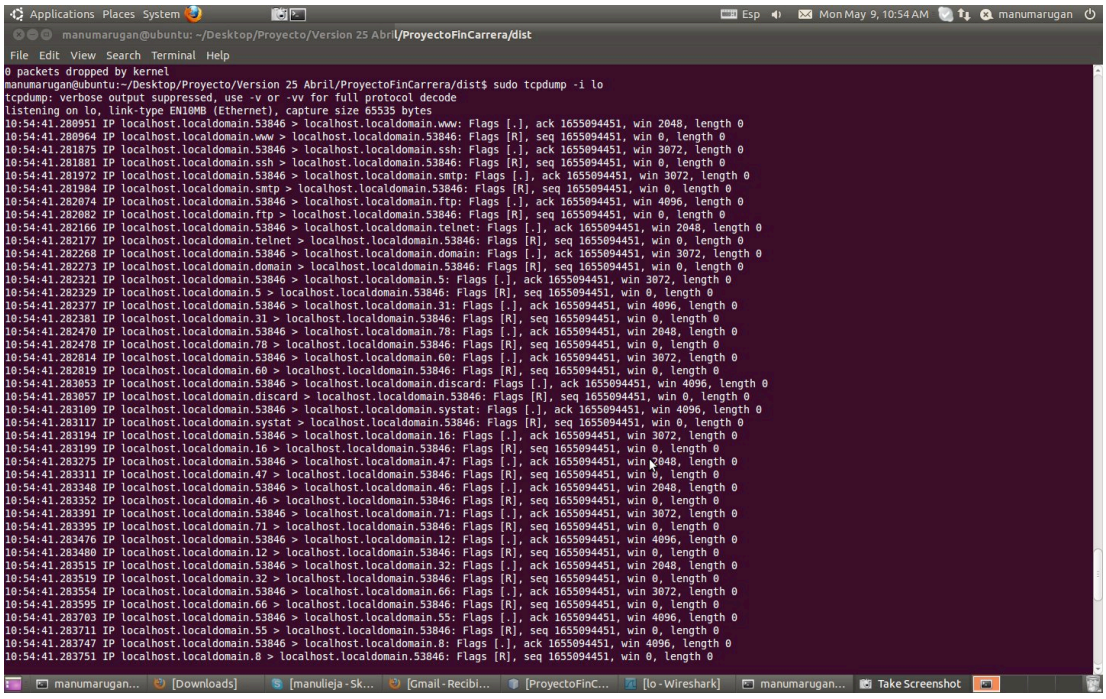


Figura 16: Tráfico de la red con Tcpdump

5.5 ECLIPSE

Eclipse ha constituido la herramienta de desarrollo de nuestra aplicación. La principal razón por la cual escogimos *Eclipse* fue porque es la herramienta que hemos usado durante estos últimos años y en la cual más aplicaciones hemos desarrollado.

Eclipse es un entorno de desarrollo integrado (IDE*, Integrated Development Environment*), es decir, un programa compuesto por un conjunto de herramientas que facilitan la labor del programador a la hora de crear sus aplicaciones. Además, es multiplataforma y de código abierto.

Como todo IDE, consta de un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI). *Eclipse* además cuenta con las siguientes herramientas: control de versiones CVS o *Subversion*, resaltado y autocompletado de sintaxis, compilación automática, asistentes para la creación de proyectos, clases, tests, etc.

Ejemplo práctico:

En la siguiente imagen podemos observar una captura de pantalla de nuestra aplicación durante su proceso de desarrollo mediante la herramienta *Eclipse*.

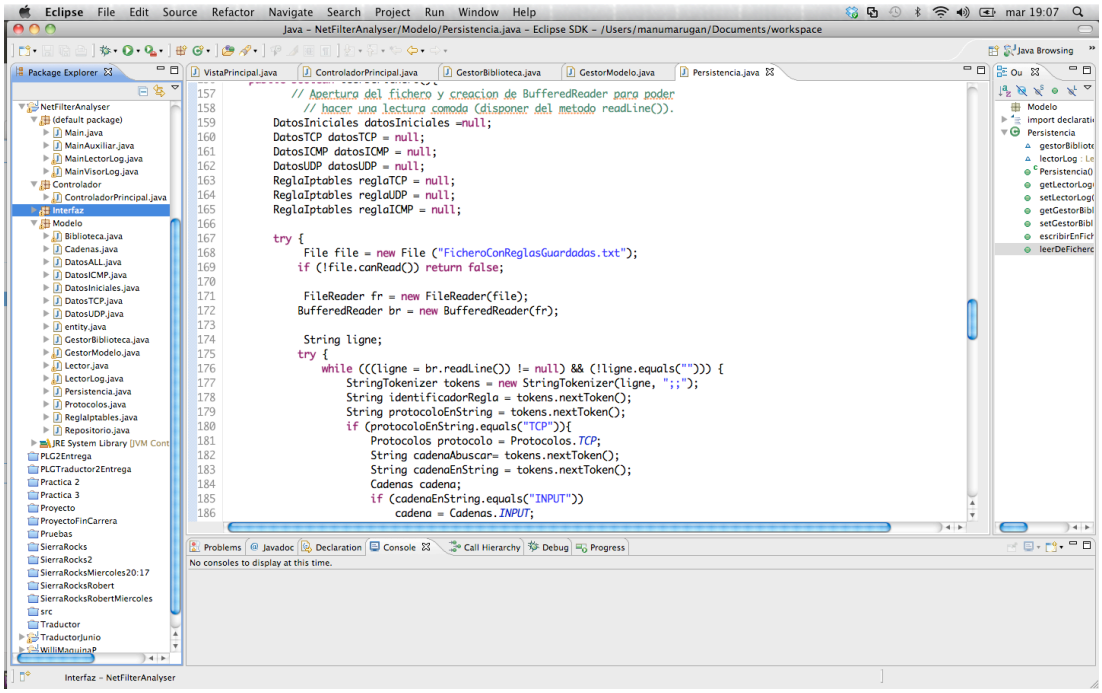


Figura 17: Editor Eclipse

5.6 SKYPE

La razón principal por la que escogimos tratar de estudiar el funcionamiento de *Skype* fue debido a su elevado porcentaje de uso como sistema de comunicación a través de Internet. *Skype* es un *software* que permite realizar comunicaciones tanto de texto, voz y video utilizando la tecnología ya mencionada VoIP.

Sus creadores *Jaanus Friis* y *Niklas Zennstrom* fueron también los creadores del programa *Kaaza*. De hecho, la gran diferencia entre *Skype* y otros sistemas de comunicación por VoIP es que *Skype* opera en base al modelo P2P en vez del usual modelo Cliente-Servidor.

El gran éxito de *Skype* reside en la gran compresión de datos que realiza, sin afectar prácticamente a la calidad de la transmisión de voz. *Skype* se jacta de proporcionar una mayor calidad de voz que *MSN* y *Yahoo!*. A su vez, es extremadamente fácil de instalar y utilizar.

El programa ha sido desarrollado en lenguaje *Pascal* usando el entorno *Delphi*.

Por otro lado, cabe destacar que *Skype* es multiplataforma. A continuación, se muestran algunos de los sistemas operativos compatibles:

- *Windows 2000, Windows XP, Windows Vista, Windows 7, Windows Mobile.*
- *Mac OS X.*
- *iOS.*
- *Android.*
- *GNU/Linux.*
- *PlayStation Portable.*
- *Symbian s60 5th edition.*

A continuación, se muestra una imagen de dicha herramienta en funcionamiento:

Figura 18: Herramienta de comunicaciones Skype

Protocolo

Las especificaciones del protocolo con el que trabaja la aplicación *Skype* no han sido jamás hecho públicas ya que se trata de un protocolo propietario. A pesar de ello, ha habido muchos intentos de hacerse con el protocolo de *Skype*. Recientemente, un desarrollador *freelance,* de nombre *Efim Bushmanov*, utilizando ingeniería inversa**,** ha conseguido descifrar buena parte del código fuente del protocolo de *Skype* y lo ha compartido en su blog: *http://skype-open-source.blogspot.com.* Finalmente, *Efim Bushmanov* ha sido denunciado y obligado a retirar el código.

Como nota de interés, debemos decir que estos programas están siendo muy apreciados por la gran masa social y suponen una revolución en las telecomunicaciones de tal modo que las grandes empresas informáticas se interesan por ellas tal y como ha sido el caso de *Microsoft* y *Skype*:

> *"Microsoft ha comprado la empresa de telefonía por Internet Skype por 8.500 millones de dólares (5.920 millones de euros), cantidad que incluye la deuda que arrastraba Skype, de unos 1.000 millones."*
>
> *El país, 10.05.2011.*

5.7 GOOGLE TALK

Otra de las aplicaciones a estudiar enmarcadas dentro del ámbito de las comunicaciones a través de Internet ha sido *Google Talk*.

Google Talk es un cliente de mensajería instantánea y VoIP de protocolo XMPP, desarrollado por *Google*. Dicha aplicación está disponible para el sistema operativo *Microsoft Windows* y para el sistema operativo de dispositivos móviles *Android*.

Resulta necesario recalcar la diferencia entre el *plugin* de *Google* para *Gmail* y el *software* de *Google Talk*. El *plugin* es un complemento de voz y video que se instala en el navegador proporcionando casi todas las funcionalidades que la aplicación *Google Talk* proporciona y además está disponible tanto para *Windows, Mac* o *Linux*.

El protocolo utilizado en *Google Talk* para el intercambio de mensajería instantánea es el protocolo abierto XMPP (*eXtensible Messaging and Presence Protocol*) originalmente llamado *Jabber*. Dicho protocolo es extensible y basado en XML. A diferencia de los protocolos propietarios, éste se encuentra muy bien documentado y accesible. De hecho, el desarrollo de esta tecnología no está ligado a ninguna empresa en concreto y no requiere de pagos.

Google diseñó una extensión al protocolo XMPP para el flujo de audio mediante RTP denominado *Jingle*. Dicha extensión implementa la transferencia de información *peer to peer* para permitir la comunicación por VoIP. *Jingle* fue liberado inicialmente, de tal forma que cualquier cliente lo puede incluir. Además, el protocolo se ha diseñado intentando mantener cierta compatibilidad con SIP.

A continuación, podemos apreciar una imagen en la que figuran los integrantes del grupo haciendo uso de dicha herramienta para su posterior estudio.

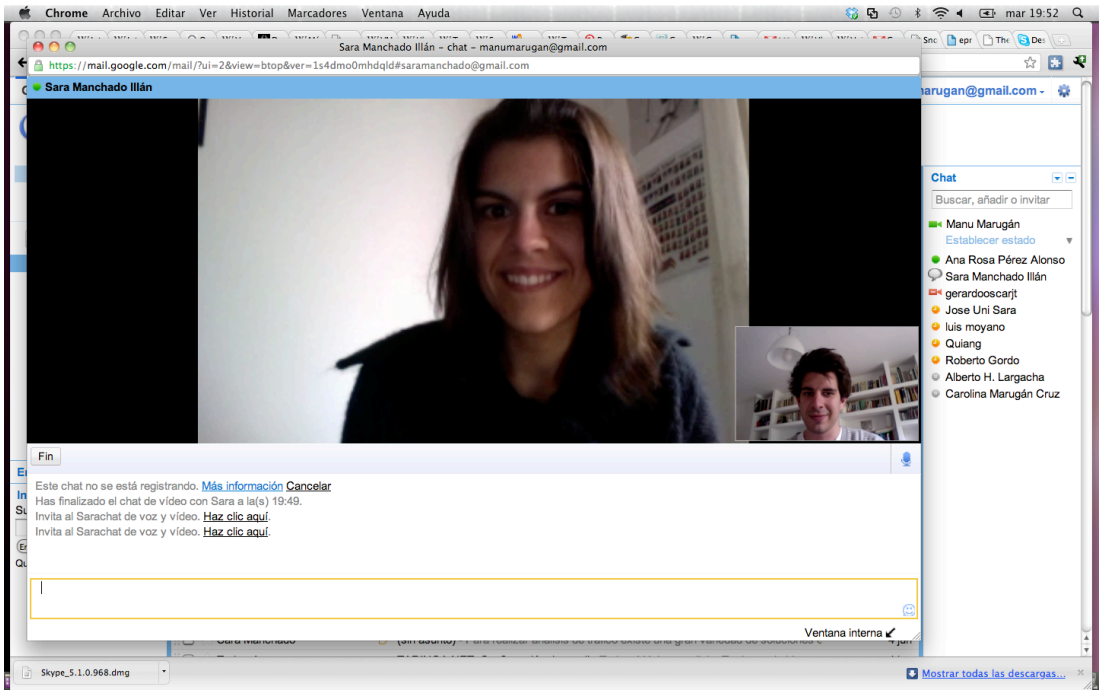


Figura 19: Herramienta de comunicaciones Google Talk

Capítulo 6.

6. SNORT

6.1 INTRODUCCIÓN

Snort es un sistema de detección de intrusos (IDS) de código abierto que puede descargarse en la página oficial de *Snort* (*http://www.snort.org*).

Analiza en tiempo real el tráfico de una red, así como registra, alerta y actúa ante cualquier anomalía previamente definida.

Por otro lado, existe una amplia gama de herramientas que pueden utilizarse con *Snort* para ampliar sus funcionalidades y con ello convertirse en una herramienta más potente.

Snort utiliza un lenguaje de reglas para especificar qué tráfico de red debe alertarse y/o tratarse en caso de ser detectado.

Entre sus ventajas destacan:

- *Software* libre.
- Se puede ejecutar en múltiples plataformas.
- Constante actualización.
- Dispone de una amplia documentación.
- Fácilmente configurable.

6.2 COMPONENTES DE SNORT

Los componentes de la arquitectura de *Snort* son:

- **Módulo de captura del tráfico de red**. Su función es capturar todos los paquetes de la Red haciendo uso de la librería *Libpcap*.

- **Decodificador**. Es el encargado de crear las estructuras de datos con los paquetes capturados e identificar los protocolos.

- **Preprocesadores**. Permiten extender las funcionalidades preparando los datos para la detección. Existen diferentes tipos de preprocesadores dependiendo del tráfico que queremos detectar.

- **Motor de Detección**. Su función es analizar los paquetes de la Red basándose en las reglas definidas para detectar comportamientos anómalos o sospechosos.

- **Archivo de Reglas**. Definen el conjunto de reglas para el análisis de los paquetes capturados.

- ***Plugins* de detección**. Son partes del *software* que son compilados con *Snort* y se usan para modificar el motor de detección.

- ***Plugins* de salida**. Permiten definir qué, cómo y dónde se guardan las alertas y los correspondientes paquetes de red que las generaron. Pueden ser archivos de texto, bases de datos, *syslog*, entre otros.

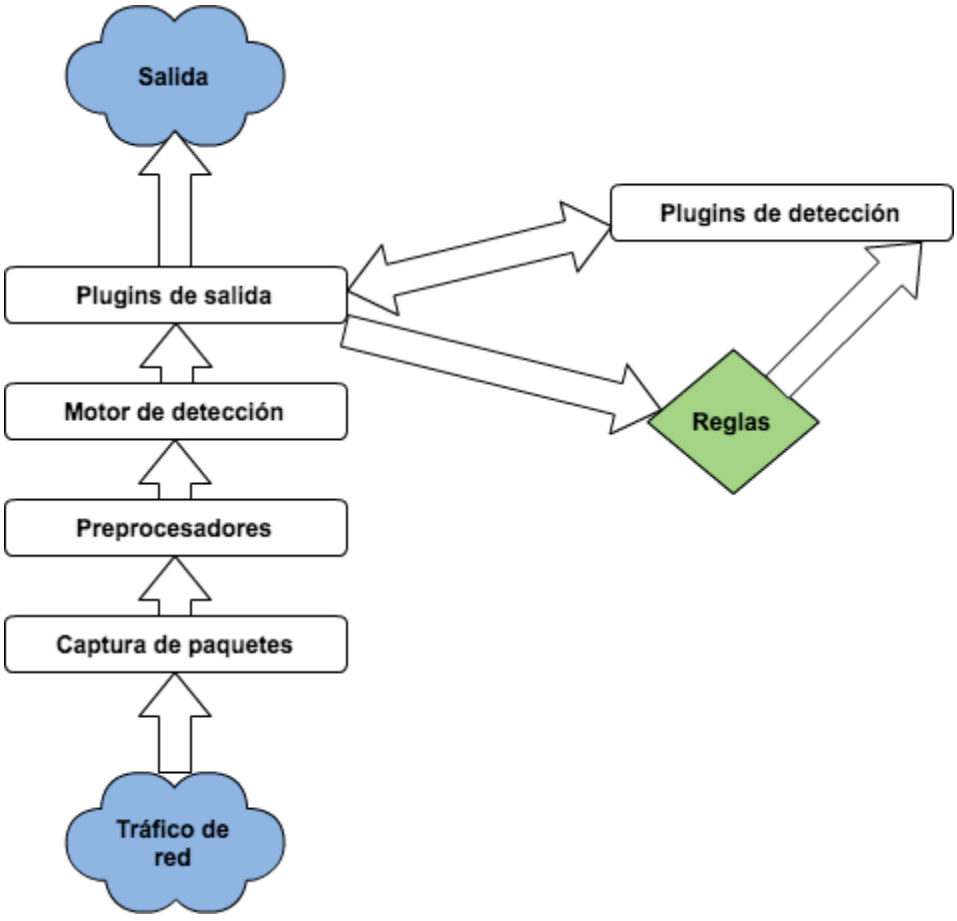


Figura 20: Arquitectura de Snort

6.2.1 MÓDULO DE CAPTURA DE DATOS

Este módulo tiene la finalidad, como su propio nombre indica, de realizar la captura del tráfico que circula por la Red, aprovechando al máximo los recursos de procesamiento y minimizando, por tanto, la pérdida de paquetes a grandes niveles.

Es necesario realizar algunas tareas previas para que los preprocesadores y posteriormente el motor de detección obtenga los paquetes de la Red. *Snort* requiere de una librería de captura de paquetes externa: *Libpcap*. *Libpcap* fue escogida para la captura de paquetes por su independencia de plataforma.

La utilización de *Libpcap* hace que *Snort* tenga un uso realmente independiente de plataforma.

6.2.2 DECODIFICADOR

La función de dicho módulo es coger paquetes que provienen de *Libpcap* y almacenarlos en una estructura de datos.

En el momento en que los paquetes sean capturados, *Snort* descifrará los elementos del protocolo correspondiente a cada paquete. Dicho decodificador está compuesto por un subconjunto de decodificadores. Cada uno de estos se encarga de descifrar los elementos de protocolos específicos. Además, dicho decodificador funciona sobre una pila de protocolos de Red semejante a la que representa la figura posterior.

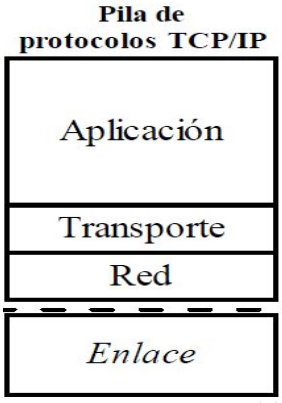


Figura 21: Pila de protocolos TCP/IP

En cuanto los paquetes de datos se almacenen en una estructura de datos estarán listos para ser analizados por los preprocesadores y el motor de detección.

6.2.3 PREPROCESADORES

Su función principal es coger la información, que viaja por la Red de una manera caótica, y darle formato con el fin de mejorar la interpretación de la información de los paquetes. Una vez que tenemos los datos ordenados aplicaremos las reglas para buscar un determinado ataque.

Los preprocesadores de *Snort* son pequeños programas en *C* que toman decisiones sobre qué hacer con el paquete. Estos programas se compilan junto a *Snort* en forma de librería.

Los preprocesadores son llamados justo después que *Snort* realice la decodificación, y posteriormente se llama al motor de detección. Hay que tener cuidado con el número de preprocesadores usados, ya que podría verse afectado el rendimiento de *Snor*t.

6.2.4 MOTOR DE DETECCIÓN

El motor de detección es el módulo más importante de *Snort*. Su principal responsabilidad es detectar cualquier actividad de intrusión existente en un paquete. Para ello, el motor de detección hace uso de las reglas de *Snort*. Las reglas son leídas en estructuras de datos internas o cadenas donde son comparadas con cada paquete. En caso de que un paquete encaje con una o varias reglas, se realiza la acción asociada. De lo contrario el paquete es descartado del análisis. Las acciones asociadas pueden ser registrar el paquete o generar alarmas.

El tiempo de análisis es un factor crítico de *Snort*. Los principales factores que influyen en el tiempo de respuesta y en la carga del motor de detección son los siguientes:

- La potencia de la máquina.
- La cantidad de reglas definidas.
- Velocidad interna del bus usado en la máquina *Snort*.
- Carga en la red.

Estos factores son muy importantes debido a que si el tráfico en la Red es demasiado alto, mientras *Snort* está funcionando en modo IDS, se podrían descartar paquetes y no se conseguiría una respuesta en tiempo real. Así pues, este es un dato importante a tener en cuenta en nuestra aplicación ya que nosotros no podremos controlar el tráfico de la Red ya que depende de la infraestructura en la que se instale la aplicación.

El motor de detección de *Snort* funciona de forma diferente en distintas versiones de *Snort*.

6.2.5 MÓDULOS DE SALIDA

Los módulos de salida, también llamados *plugins* de salida, pueden desempeñar diferentes operaciones de envío o registro de alertas dependiendo de cómo se desee guardar la salida generada por el sistema. Básicamente estos módulos controlan el tipo de salida generada por estos sistemas.

Existen varios módulos de salida que se pueden utilizar, dependiendo del formato en el que se deseen los datos: *syslog*, *database* y el nuevo módulo denominado *unified*, que es un formato binario genérico para exportar datos a otros programas.

Tipos de módulos de salidas más comunes:

Syslog. Envío de alarmas al *syslog*.

Alert_Fast. El módulo alerta rápida devuelve información sobre: tiempo, mensaje de la alerta, clasificación, prioridad de la alerta, direcciones IP y puerto de origen y destino.

Alert_Full. El módulo alerta completa devuelve información sobre: tiempo, mensaje de la alerta, clasificación, prioridad de la alerta, direcciones IP, puerto de origen y destino e información completa de las cabeceras de los paquetes registrados.

Alert_smb. Permite realizar llamadas al cliente de SMB, y enviar mensajes de alerta a un host *Windows*.

Alert_unixsock. Este módulo manda las alertas a través de un *socket*, para que las escuche otra aplicación.

Log_tcpdump. Este módulo asocia paquetes a un archivo con formato *Tcpdump*.

Database. *Snort* soporta directamente cuatro tipos de salida a base de datos: *MySQL*, *PostgreSQL*, *Oracle* y *unixODBC.*

Unified. Es un formato binario básico para registrar los datos y usarlos en el futuro. Los dos argumentos admitidos son *filename* y *limit*.

Log Null. Este módulo genera reglas que provocarán alertas sobre ciertos tipos de tráfico, pero no causarán entradas en los archivos del *Log*.

6.2.6 PLUGINS DE SNORT

Hay una serie de *plugins* para *Snort*, que contribuyen a aumentar su funcionalidad. Alguno de ellos son: *Spade, InlineSnort, BRO* o *SAM*.

6.3 REGLAS

Las reglas, también denominadas firmas, definen los perfiles que se buscan dentro de los paquetes de datos. Las reglas de *Snort* son utilizadas por el motor de detección para comparar los paquetes recibidos y generar las alertas en caso de existir coincidencia entre el contenido de los paquetes y las reglas.

Las reglas pueden ser creadas "a mano" por el usuario o se puede hacer uso de un conjunto de reglas predefinidas que vienen adjuntas al descargarse la aplicación.

En el archivo "*snort.conf*" se guarda toda la información de la configuración de las reglas, preprocesadores y otras configuraciones necesarias para el correcto funcionamiento de la herramienta. En la parte final del archivo se pueden ver todos los conjuntos de reglas de alertas. Se puede desactivar toda una categoría de reglas comentando la línea de la misma.

A continuación, se detallan las principales características de las reglas de *Snort*.

6.3.1 CATEGORÍAS DE REGLAS

Hay cuatro categorías de reglas para evaluar un paquete:

Reglas de Protocolo. Son aquellas reglas que dependen del protocolo que se está analizando.

Reglas de Contenido Genéricas. Son reglas que permiten especificar patrones para buscar en el campo de datos del paquete, los patrones de búsqueda pueden ser binarios o en modo ASCII.

Reglas de Paquetes Malformados. Son reglas que especifican características sobre los paquetes, concretamente sobre sus cabeceras las cuales indican que se está produciendo algún tipo de anomalía. Este tipo de reglas no miran en el contenido ya

que primero se comprueban las cabeceras en busca de incoherencias u otro tipo de anomalía.

Reglas IP. Reglas que se aplican directamente sobre la capa IP, y son comprobadas para cada datagrama IP. Si el datagrama contiene TCP, UDP o ICMP, se realizará un análisis del datagrama con su correspondiente capa de protocolo. Este tipo de reglas permite analizar con contenido y sin él.

6.3.2 ESTRUCTURA DE LAS REGLAS

La mayoría de las reglas de *Snort* se escriben en una sola línea. Sin embargo, cabe la posibilidad de escribir una regla en varias líneas añadiendo una barra inversa (\) al final de la línea.

Una regla de *Snort* consta de dos secciones básicas:

- Cabecera
- Opciones

6.3.2.1 Cabecera de una regla

La cabecera de una regla contiene información sobre: la acción de la regla, el protocolo, las direcciones IP fuente y destino y máscaras de red, e información del puerto origen y destino. Su estructura es la siguiente:

<acción><protocolo><redOrigen><puertoOrigen><dirección><redDestino>
<puertoDestino>

Y el significado de cada campo es el siguiente:

Protocolo. Indica el protocolo que se va a utilizar. Los posibles valores son: TCP, UDP, IP e ICMP.

Red de origen y red de destino. Permite establecer el origen y el destino de la comunicación.

Puerto de origen y destino. Permite establecer los puertos origen y destino de la comunicación.

Dirección. Indica el sentido de la comunicación. Las posibles opciones son: ->, <- y <>.

Acción. Permite indicar la acción que se debe realizar sobre dicho paquete. Los posibles valores son:

- ***alert*:** genera una alerta para posteriormente registrar el paquete.
- ***log*:** registra el paquete.
- ***pass*:** ignora el paquete.
- ***activate*:** alerta y luego activa otra regla dinámica.
- ***dynamic*:** permanece ocioso hasta que se active por otra regla.

6.3.2.2 Opciones de una regla

La sección de opciones va encerrada entre paréntesis y cada una de las opciones están separadas entre sí, por (;) y las claves de las opciones están separadas por (:). Hay cuatro tipos de opciones:

- ***Metadata*.** Permite integrar información adicional sobre la regla.
- ***Payload*.** Busca contenido dentro de la carga útil del paquete.
- ***Non-Payload*.** Busca contenido dentro de los demás campos del paquete que no sean carga útil (por ejemplo, la cabecera).
- ***Post-detection*.** Permite activar reglas específicas que ocurren después de que se ejecute una regla.

A continuación, se describen las principales opciones de las reglas:

msg. Define el mensaje a mostrar en el *Log*. Los caracteres especiales de las reglas como (:) y (;) deben colocarse dentro de la opción msg con el carácter (\).

flow. Indica las reglas que deben contrastarse con ciertos tipos de tráfico.

content. Permite que *Snort* realice una búsqueda en el contenido de datos de un paquete. Es sensible a mayúsculas y minúsculas.

classtype. Indica el tipo de ataque. La opción *classtype*, usa las *classifications* definidas en el archivo de configuración de *Snort* y que se encuentra en el fichero "*classification.config"*.

La sintaxis del *classification.config* es:
<nombreClase>, <descripciónClase>, <prioridadPorDefecto>.

La prioridad es un valor entero, normalmente para indicar una prioridad alta se pone el valor uno, para media dos y tres para prioridad baja.

Rev. Revisión de la regla por posibles futuras modificaciones.

Sid. Se usa en combinación con la opción *rev*, únicamente identifica una regla *Snort*, correlacionando el ID de la regla individual con la revisión de la regla.

6.3.3 TRABAJANDO CON SNORT

En la siguiente figura se muestra un listado de reglas creadas por nosotros durante el estudio de *Snort*. Las reglas están guardadas en un fichero cuyo nombre es "*sara.rules*".

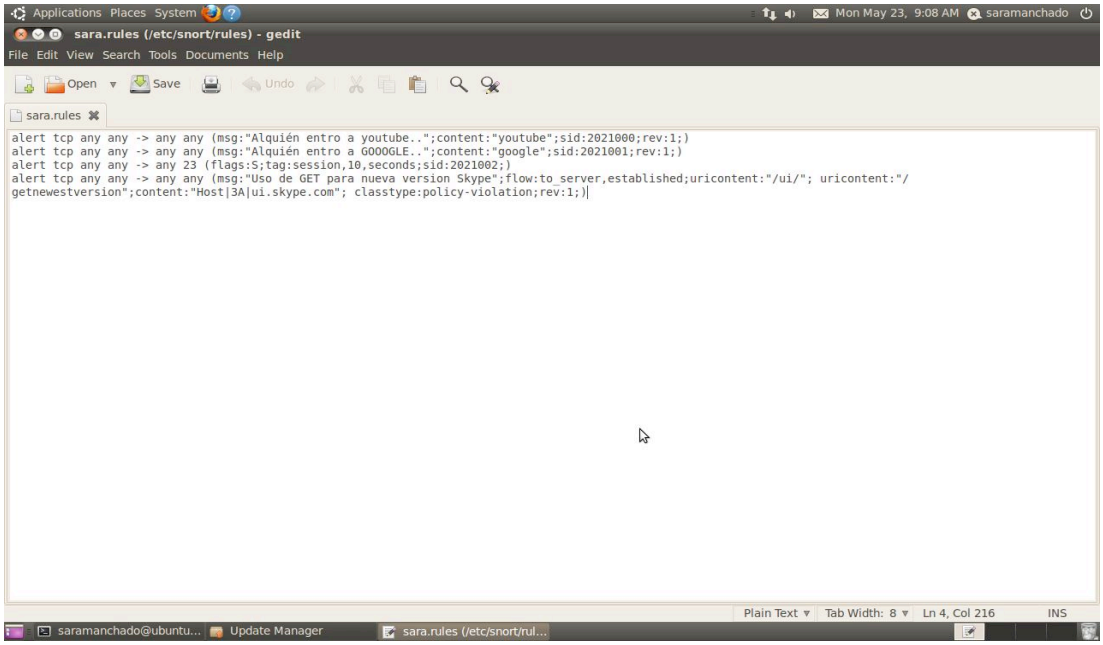


Figura 22: Listado de reglas en el archivo "sara.rules"

A continuación, es necesario modificar el fichero "*snort.conf*" con el objetivo de indicar las reglas que queremos utilizar en la detección de tráfico. Para ello, debemos añadir en "*snort.conf*" la línea: *include $RULE_PATH/sara.rules*

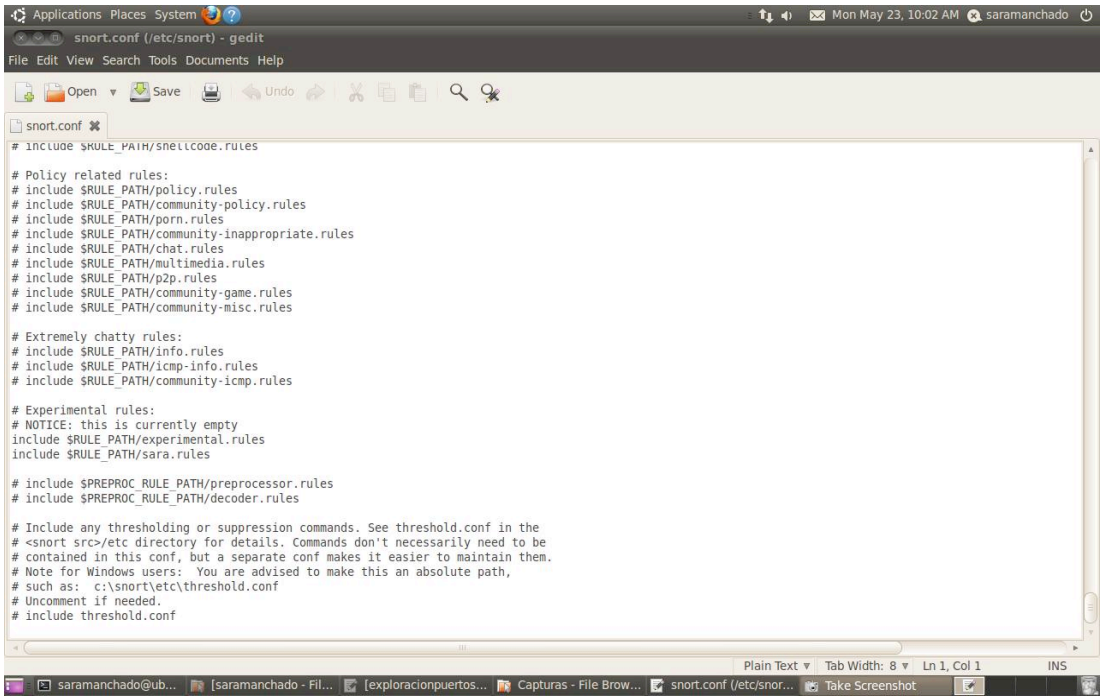


Figura 23: Configuración del archivo "snort.conf"

El siguiente paso es lanzar *Snort* con el siguiente comando:

sudo snort –v –ieth0

Como podemos observar en la siguiente figura, *Snort* arranca y se queda activo procesando el tráfico y alertando en caso de que alguna de las reglas de *Snort* se active.

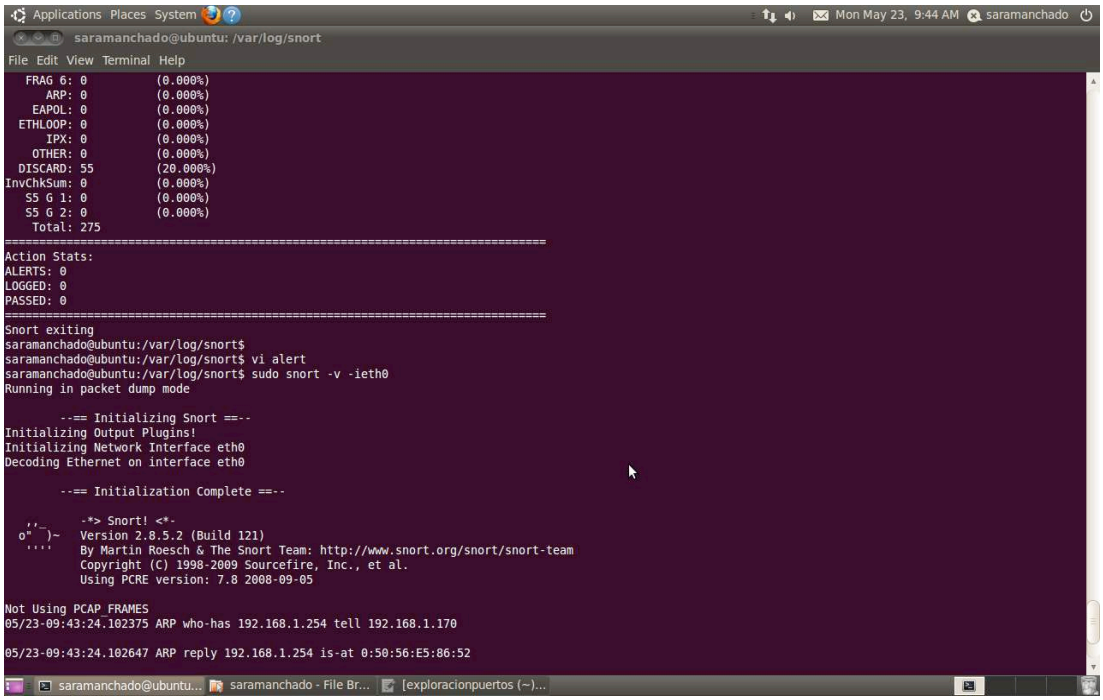


Figura 24: Ejecución de Snort

En las siguientes figuras podemos observar alertas lanzadas por *Snort* como consecuencia de la activación de nuestras reglas.

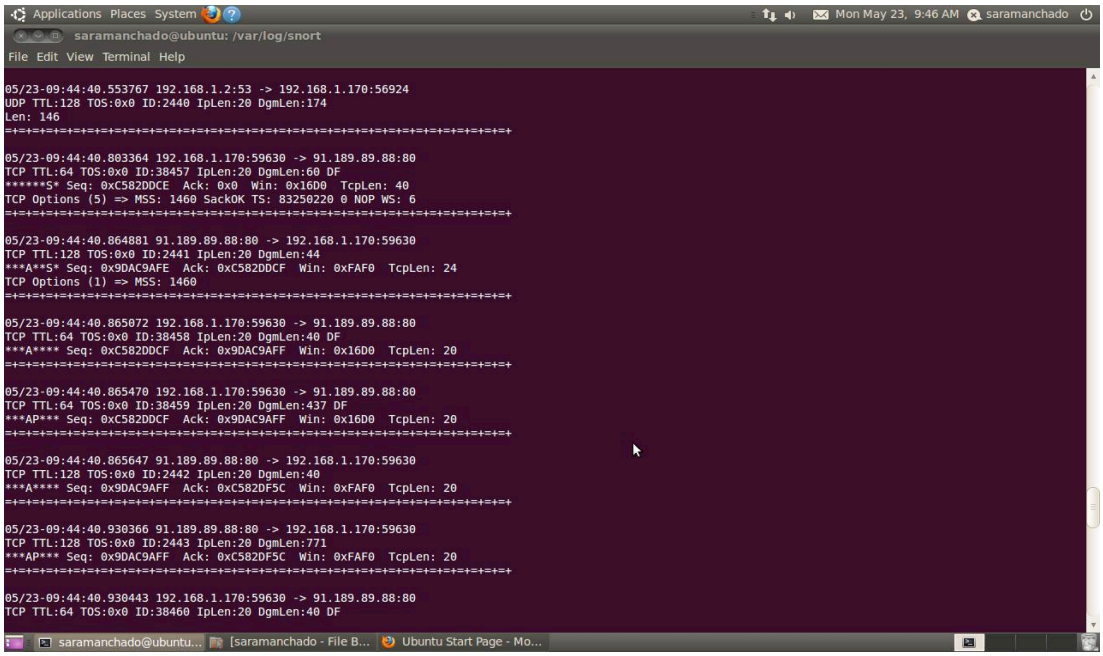


Figura 25: Alertas de Snort (Parte 1)

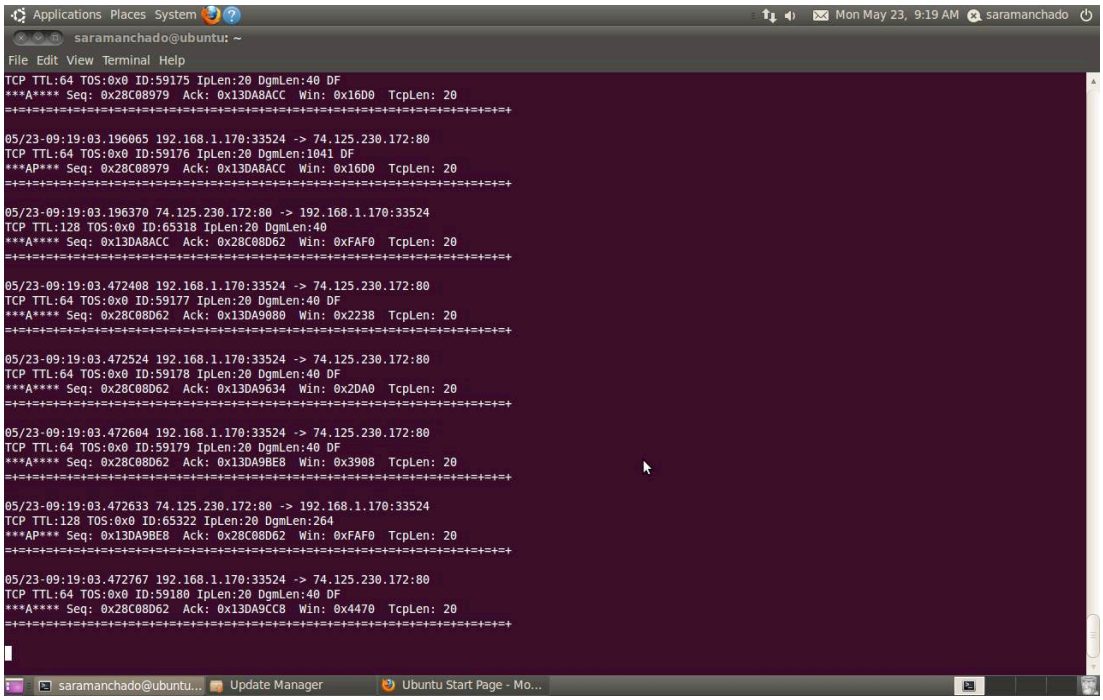


Figura 26: Alertas de Snort (Parte 2)

Capítulo 7.

7. IPTABLES

7.1 INTRODUCCIÓN

Iptables es un *firewall* que permite filtrar paquetes, realizar traducción de direcciones de red (NAT) para IPv4 o mantener registros del *Log*. *Iptables* está vinculado al *kernel* de Linux. Se trata de una herramienta muy compleja, y tremendamente configurable.

Iptables permite definir reglas para controlar el tráfico de red. Requiere privilegios de administrador del sistema para operar, por ello, el único modo de ejecutarlo es en modo superusuario.

7.2 REGLAS DE IPTABLES

Las reglas se agrupan en cadenas, las cuales a su vez están agrupadas en tablas. A continuación, se expone información más detallada acerca de las mismas.

Cadenas

Una cadena es una lista ordenada de reglas. Cuando recogemos un paquete se envía a una cadena y se compara, en orden, con cada una de las reglas de la cadena. La regla define qué características debe tener el paquete para que la regla se active, tales como tipo de protocolo, dirección IP origen/destino o número de puerto. Tras la comparación, la regla puede haber sido activada o no. En caso de que no se haya activado la regla el proceso continúa con la siguiente regla. Si el paquete, por el contrario, coincide con la regla, las especificaciones indicadas en la regla para tratar el paquete se siguen. Por lo general, cualquier otro procesamiento de la cadena normalmente se aborta.

Tablas

En *Iptables* se definen tres tablas principales: *Filter, Nat* y *Mangle*.

Tabla Filter. Esta tabla es la encargada de filtrar los paquetes. Se comparan los paquetes con las condiciones indicadas en las reglas y en función del resultado se

realiza una acción como aceptar el paquete o descartarlo. La tabla *Filter* está compuesta por cadenas predefinidas:

- INPUT. Todos los paquetes entrantes y dirigidos al propio sistema deberán atravesar esta cadena.
- OUTPUT. Todos los paquetes que salen y han sido originados en el propio sistema deberán atravesar esta cadena.
- FORWARD. Todos los paquetes que pasan por el sistema para su encaminamiento hacia su destino deberán atravesar esta cadena.

Tabla Nat. Esta tabla es la encargada de configurar el protocolo NAT (*Network Address Translation*) sobre los paquetes. Se usa para la traducción de los campos: dirección IP origen y dirección IP destino del paquete. Dependiendo del campo que se desee modificar se emplea una de las siguientes cadenas:

- PREROUTING. Esta cadena se usa principalmente para la traducción de direcciones de red de destino (DNAT, *Destination Network Address Translation*).
- POSTROUTING. Esta cadena se usa para la traducción de direcciones de red de origen (SNAT, *Source Network Address Translation*).
- OUTPUT. Permite hacer NAT a los paquetes que salen desde el host.

Tabla Mangle. Esta tabla es la encargada de modificar el contenido de los paquetes. La tabla *Mangle* está formada por las cadenas predefinidas: PREROUTING, INPUT, FORWARD, OUTPUT y POSTROUTING.

7.3 FUNCIONAMIENTO DE IPTABLES

Un paquete entra por una interfaz de red como puede ser una tarjeta de red o módem. El paquete se dirige al *firewall*. Éste empieza a procesarlo. El paquete atraviesa las tablas de *Iptables* con sus correspondientes cadenas. El paquete, irá pasando secuencialmente por cada una de las reglas de la cadena hasta coincidir con el patrón de alguna de ellas. Cuando esto ocurra, el paquete se tratará según indique la acción de la regla. Si tras recorrer toda la lista, el paquete no coincide con ninguna de las reglas, se ejecutará la acción por defecto asociada a esa cadena.

En la figura siguiente se detalla de forma gráfica las diferentes rutas posibles que puede seguir un paquete durante su procesado a través del *firewall*.

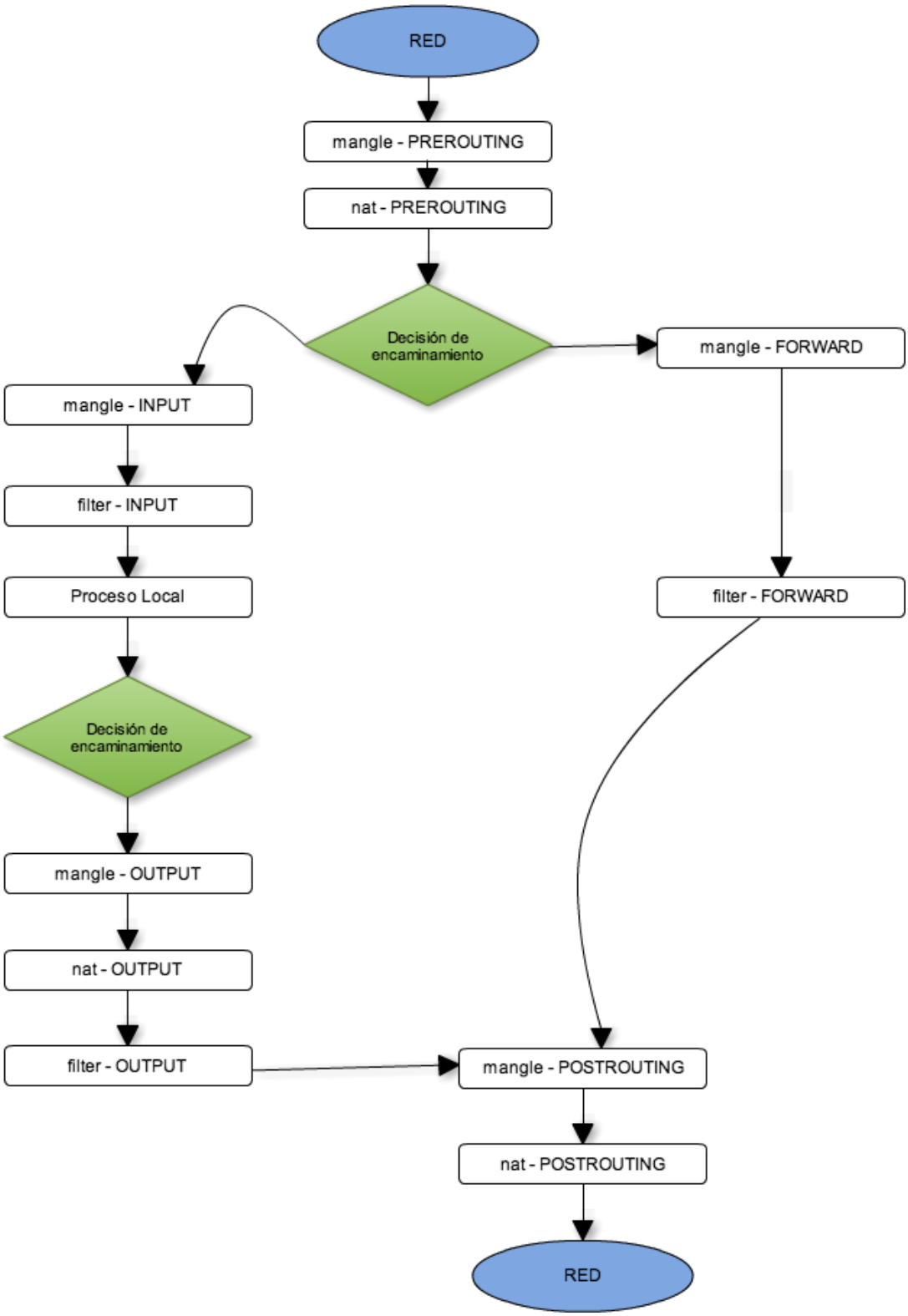


Figura 27: Procesado de paquetes en Iptables

7.4 EL COMANDO "IPTABLES"

El objetivo de este apartado es explicar el uso del comando "*iptables*" para crear y gestionar las reglas de nuestro cortafuegos. A pesar de la descripción que vamos a ofrecer, si se desea un conocimiento exhaustivo de todas las opciones de la herramienta *Iptables* recomendamos consultar el manual de *Iptables*.

En primer lugar, veremos cómo gestionar las cadenas. La utilización del comando "*iptables*" presenta siempre la siguiente estructura:

iptables [<*-t tabla*>] *comando* <*match*><*objetivos/saltos*>

Las tablas son siempre una de las tres definidas anteriormente: *Filter, Nat*, o *Mangle.* En caso de no indicar ninguna en concreto, por defecto, nos referimos a la tabla *Filter*.

Para añadir y manipular cadenas utilizaremos los comandos siguientes:

iptables –N: crea una nueva cadena vacía, es decir, sin reglas.
iptables –X: elimina una cadena que esté vacía.
iptables –F: vacía una cadena, es decir, elimina todas las reglas de una cadena.
iptables –P: cambia la política por defecto de una cadena.
iptables –L: lista las reglas de una cadena.
iptables –Z: pone a cero las contadores de una regla (número de paquetes, de *bytes*, etc.).

Con los comandos siguientes conseguimos gestionar las reglas de esas cadenas:

iptables –A: añade al final de una cadena una nueva regla.
iptables –I: inserta al comienzo de una cadena una nueva regla.
iptables –R: reemplaza una regla de una cadena.
iptables –D: elimina una regla de una cadena.

Por último, vamos a explicar cómo podemos establecer las condiciones y acciones sobre cada regla, es decir, cómo establecer las condiciones de comparación entre paquetes y reglas:

–s: indica la dirección de origen sobre el que se evalúa la condición de la regla.
–d: indica la dirección de destino sobre el que se evalúa la condición de la regla.
–i: indica la interfaz de entrada sobre la cual se evalúa la condición de la regla.
–o: indica la interfaz de salida sobre la cual se evalúa la condición de la regla.

–p: indica el protocolo del datagrama que concordará con esta regla. Los protocolos válidos son TCP, UDP, ICMP, o un número, en caso de conocer el número del protocolo IP.

Cada protocolo lleva asociados sus propios campos:

Campos de TCP:

- *--sport*: indica el puerto origen que debe contener el segmento para que se active la regla. Se pueden especificar los puertos en la forma de un rango, especificando los extremos inferior y superior usando los dos puntos (:) como delimitador.
- *--dport*: indica el puerto destino que debe contener el segmento para que se active la regla. Se pueden especificar los puertos en la forma de un rango, especificando los extremos inferior y superior usando los dos puntos (:) como delimitador.
- *--tcp -flags*: especifica mediante una máscara los bits indicadores de TCP del datagrama. La máscara está compuesta por *flags* separados por comas entre sí. Los *flags* son: SYN, ACK, FIN, RST, URG, PSH, ALL o NONE.

Campos UDP:

- *--sport*: indica el puerto origen que debe contener el datagrama para que se active la regla.
- *--dport*: indica el puerto destino que debe contener el datagrama para que se active la regla.

Campos ICMP:

- *--icmp-type*: especifica el tipo de mensaje ICMP. Se puede especificar mediante su número asociado o por los siguientes identificadores: *echo-request, echo-reply, source-quench, time-exceeded, destination-unreachable, network-unreachable, host-unreachable, protocol-unreachable* y *port-unreachable*.

Campos MAC:

- *--mac -source*: especifica la dirección MAC. Este campo sólo tiene sentido en la cadena INPUT y FORWARD.

–f: esta opción se usa cuando se fracciona un datagrama porque supera el MTU de la Red. Permite especificar acciones sobre el segundo y restantes fragmentos del datagrama.
! : invierte el valor lógico de la condición de la regla.

Existen más posibilidades de filtrado, están son las que más hemos utilizado durante el desarrollo del proyecto para la detección de comportamientos de la Red.

7.5 TRABAJANDO CON IPTABLES

A continuación se muestran algunas de las órdenes más habituales durante el uso de la herramienta *Iptables*.

Para listar todas las reglas que hay en *Iptables*, introducimos en la terminal el siguiente comando:

sudo iptables –L –n

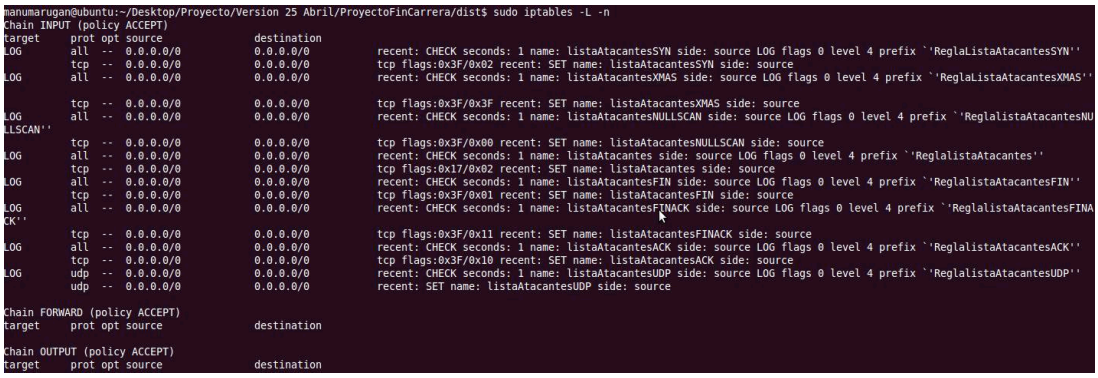


Figura 28: Listar reglas en Iptables

Para insertar una regla en *Iptables*, introducimos en la terminal el siguiente comando:

sudo iptables –A INPUT –p TCP –j LOG --log-prefix ´reglaTCP´

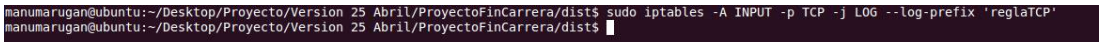


Figura 29: Insertar una regla en Iptables

Para borrar una regla de *Iptables*, introducimos en la terminal el siguiente comando:

sudo iptables –D INPUT –p TCP –j LOG --log-prefix ´reglaTCP´

```
manumarugan@ubuntu:~/Desktop/Proyecto/Version 25 Abril/ProyectoFinCarrera/dist$ sudo iptables -A INPUT -p TCP -j LOG --log-prefix 'reglaTCP'
manumarugan@ubuntu:~/Desktop/Proyecto/Version 25 Abril/ProyectoFinCarrera/dist$ sudo iptables -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
LOG        tcp  --  0.0.0.0/0            0.0.0.0/0           LOG flags 0 level 4 prefix `reglaTCP'

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
manumarugan@ubuntu:~/Desktop/Proyecto/Version 25 Abril/ProyectoFinCarrera/dist$ sudo iptables -D INPUT -p TCP -j LOG --log-prefix 'reglaTCP'
manumarugan@ubuntu:~/Desktop/Proyecto/Version 25 Abril/ProyectoFinCarrera/dist$
```

Figura 30: Borrar una regla en Iptables

Listamos de nuevo las reglas de *Iptables* y comprobamos que la regla ha sido eliminada correctamente.

```
manumarugan@ubuntu:~/Desktop/Proyecto/Version 25 Abril/ProyectoFinCarrera/dist$ sudo iptables -D INPUT -p TCP -j LOG --log-prefix 'reglaTCP'
manumarugan@ubuntu:~/Desktop/Proyecto/Version 25 Abril/ProyectoFinCarrera/dist$ sudo iptables -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
manumarugan@ubuntu:~/Desktop/Proyecto/Version 25 Abril/ProyectoFinCarrera/dist$
```

Figura 31: Listar reglas en Iptables

Para borrar todas las reglas de *Iptables*, introducimos en la terminal el siguiente comando:

sudo iptables –F

```
manumarugan@ubuntu:~/Desktop/Proyecto/Version 25 Abril/ProyectoFinCarrera/dist$ sudo iptables -F
manumarugan@ubuntu:~/Desktop/Proyecto/Version 25 Abril/ProyectoFinCarrera/dist$ sudo iptables -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
manumarugan@ubuntu:~/Desktop/Proyecto/Version 25 Abril/ProyectoFinCarrera/dist$
```

Figura 32: Borrar todas las reglas en Iptables

Por otra parte, cabe destacar que *Iptables* ofrece una amplia gama de módulos para mejorar el alcance de las reglas. Algunos de estos módulos están disponibles por defecto.

A continuación, detallamos un breve análisis del módulo *String* y del módulo *Recent* , ya que han sido los más usados en nuestras reglas.

Módulo *String*

El módulo *String* busca en el contenido del paquete una determinada cadena de caracteres. Existen dos parámetros *--from* y *--to* que se utilizan para indicar el comienzo y el final donde se desea buscar los datos. En caso de omitir dichos parámetro la búsqueda se realiza en el paquete entero.

El parámetro *--algo* indica el tipo de algoritmo usado en la búsqueda de la cadena. Existen dos tipos de algoritmos: bm (*Boyer-Moore*) y kmp (*Knuth-Pratt-Morris*).

Ejemplo de uso del módulo *String*:

-m string --string cadenaBuscar --from 48 –to 60 --algo bm

Módulo *Recent*

El módulo *Recent* permite monitorizar y limitar las conexiones recientes. El funcionamiento de este módulo es muy sencillo. Añade direcciones IP a una lista, las cuales serán posteriormente comparadas con las direcciones IP de los nuevos intentos de conexión.

-m recent --name "nombreListaAtacantes" --rcheck --seconds "numSegundos"
-m recent --name "nombreListaAtacantes" --set

El parámetro *--name* indica el nombre de la lista donde se almacenarán las direcciones IP. El parámetro *--seconds* establece el intervalo de tiempo en segundos. El parámetro *--set* inserta la dirección IP origen en la lista de direcciones IP.

Capítulo 8.

8. DETECCIÓN DE TRÁFICO

8.1 INTRODUCCIÓN

El objetivo de este capítulo es detallar de forma analítica y descriptiva los pasos que nos han llevado a la detección del tráfico por medio de nuestra aplicación "*Netfilter Analyser*".

Para poder detectar tráfico es necesario caracterizarlo de algún modo o encapsularlo dentro de un patrón de tráfico. El proceso de identificación de patrones requiere de grandes dosis de observación, para encontrar semejanzas, y de abstracción, para descartar detalles irrelevantes y describir con precisión la esencia del problema. De este modo, el proceso de identificación de patrones puede verse como una ciencia empírica y de abstracción, donde a partir de los resultados se intentan obtener leyes que explican dichos resultados.

De este modo nuestra labor consistió en su primera fase de un proceso de análisis descriptivo acerca de los comportamientos del tráfico durante la cual nos dedicábamos a generar tráfico y estudiar sus características más relevantes.

Una vez concluida la primera fase decidimos caracterizar dichos comportamientos de tráfico mediante reglas en formato *Iptables*.

Según esto, la siguiente fase consistió en el diseño y construcción de nuestra aplicación que incorporaría las reglas previas y trabajaría conjuntamente con el *firewall* de iptables para detectar el tráfico deseado.

8.2 MECANISMOS PARA LA GENERACIÓN DE TRÁFICO

Como paso previo a analizar los comportamientos de red de una forma precisa decidimos crear un entorno cerrado. Este entorno cerrado tenía como objetivo aislar el tráfico interesante del ruido existente en la Red de Internet.

Para generar dicho entorno nos servimos de la herramienta *User Mode Linux* previamente analizada y comentada. Gracias a dicha herramienta pudimos crear una infraestructura apta para realizar la generación de tráfico y su posterior análisis.

Una vez aislado el entorno de trabajo del ruido exterior nos dispusimos a generar el tráfico.

Dado que nuestro primer caso de estudio fue la exploración de puertos y la denegación de servicio, la herramienta encargada de generar dicho tráfico fue *Nmap*. Dicha herramienta también ha sido comentada y analizada previamente en el capítulo 5 con el fin de asentar las bases técnicas de este proyecto. Dicho capítulo contiene alguno de los ejemplos realizados durante la fase de simulación de tráfico.

El siguiente caso de estudio fue la Telefonía sobre IP. Para generar tráfico relativo a este servicio utilizamos herramientas tales como: *Skype, Google Talk*, *Windows Live Messenger* y *Yahoo! Messenger*. Cada una de estas herramientas genera tráfico de forma distinta por lo que caracterizar dicho tráfico fue una tarea ambiciosa y compleja.

8.3 ANÁLISIS Y CARACTERIZACIÓN DEL TRÁFICO

Para realizar el análisis de tráfico existe una gran variedad de soluciones que van desde productos propietarios que incluyen *hardware* y *software*, hasta soluciones gratuitas y de código abierto comúnmente utilizadas bajo sistemas GNU/*Linux*.

Para efectos de esta práctica basaremos el análisis en dos aplicaciones gratuitas, descritas en el capítulo 5, *Wireshark* y *Tcpdump*.

Una vez que hemos sido capaces de visualizar el tráfico de una forma aislada y correcta nos dispusimos a formular preguntas que nos ayudarían a caracterizar el tráfico asociado a dichos comportamientos de red.

Entre ellas podemos destacar las siguientes cuestiones:

- ¿Posee algún paquete una combinación de *flags* sospechosa?
- ¿Proceden muchos paquetes de la misma dirección IP origen?
- ¿Van dirigidos muchos paquetes a la misma dirección IP destino?
- ¿Cuál ha sido el número de puertos explorados?
- ¿Qué volumen de tráfico se ha generado en un determinado tiempo?
- ¿Qué protocolo de red se utiliza en dicho comportamiento?
- ¿Los paquetes proceden de una dirección IP sospechosa?

Con estas y otro tipo de preguntas empezamos a crear los primeros bocetos de perfiles de conducta de dicho tráfico. A la hora de crear los perfiles de conducta es

necesario tener presentes los falsos positivos ya que nos podría llevar a resultados equívocos.

Un falso positivo es tráfico de red que aparentemente parece malicioso cuando realmente no lo es.

Por el contrario, un falso negativo es tráfico de red que aparentemente parece normal cuando realmente se está materializando un ataque.

Durante el desarrollo de nuestra aplicación tuvimos problemas con estos factores y nuestro objetivo fue siempre minimizar la confusión con la aparición de dichos falsos.

Teniendo en cuenta todo lo anterior, empezamos a elaborar las primeras reglas capaces de caracterizar los comportamientos de red indicados previamente. Algunas de éstas se muestran a continuación:

iptables -A INPUT -m recent --name listaAtacantesSYN --rcheck --seconds 10 -j LOG --log-prefix 'ReglaListaAtacantesSYN'

iptables -A INPUT -p tcp --tcp-flags ALL SYN -m recent --name listaAtacantesSYN --set"

iptables -A INPUT -m recent --name listaAtacantesXMAS --rcheck --seconds 10 -j LOG --log-prefix 'ReglaListaAtacantesXMAS'

iptables -A INPUT -p tcp --tcp-flags FIN, URG, PUSH FIN, URG, PUSH -m recent --name listaAtacantesXMAS –set

iptables -A INPUT -m recent --name listaAtacantesNULLSCAN --rcheck --seconds 10 -j LOG --log-prefix 'ReglalistaAtacantesNULLSCAN'

iptables -A INPUT -p tcp --tcp-flags ALL NONE -m recent --name listaAtacantesNULLSCAN –set

iptables -A INPUT -p udp -m recent --name listaAtacantesUDP --rcheck --seconds 10 -j LOG --log-prefix 'ReglalistaAtacantesUDP'

iptables -A INPUT -p udp -m recent --name listaAtacantesUDP –set

iptables –A INPUT -p udp -m udp --dport 5060 -j LOG --log-prefix 'ReglaVOIP1'

iptables –A INPUT -p udp -m udp --dport 4569 -j LOG --log-prefix 'ReglaVOIP2'

iptables –A INPUT -p udp -m udp --dport 5036 -j LOG --log-prefix 'ReglaVOIP3'

iptables –A INPUT –m string --string 'SIP' -j LOG --log-prefix 'ReglaSIP'

iptables -A FORWARD -p icmp --icmp-type echo-request -m limit --limit 1/s -j LOG --log-prefix 'ReglaDenegacionDeServicio'

Estas son sólo algunas de las reglas que se encuentran en el repositorio de nuestra aplicación. En este documento hemos decidido mostrar algunas de las reglas a modo divulgativo.

A continuación, explicamos a grandes rasgos las diferentes estrategias aplicadas con base en estas reglas.

Para la captura de tráfico de exploraciones de puertos decidimos utilizar el concepto de *badguy list* o lista de maleantes. Dicho concepto se basa en capturar las direcciones IP origen de paquetes que cumplan las condiciones de la regla. Cuando otro paquete llegue con la misma dirección IP y cumpla las condiciones, dicho paquete será registrado en el *Log*. Para llevar a cabo esta estrategia hemos utilizado el módulo *Recent*. Dicho módulo está explicado en el apartado 7.5.

Con respecto al tráfico proveniente de VoIP, hemos incluido distintos tipos de reglas. Algunas de ellas, simplemente se encargan de registrar el tráfico que va destinado a un determinado puerto destino. Éste no es el mejor método, puesto que basta con cambiar dicho puerto en la configuración del *software* con el que se establece la comunicación VoIP para saltarse la regla.

Otra opción es la de incluir en la regla como condición la búsqueda de la palabra reservada "SIP" ayudándonos del módulo *String*. No basta con ello, puesto que esto genera falsos positivos cuando alguien, por poner un ejemplo, escribe en un correo electrónico la palabra "SIP". Resulta necesario especificar el lugar en el que la palabra aparecerá en el paquete de datos. El funcionamiento y sintaxis del módulo *String* está explicado con anterioridad en el apartado 7.5.

Estas y otras muchas reglas son necesarias para caracterizar el tráfico de los comportamientos citados. Cuantas más reglas se añadan a la base de reglas más probabilidades tendremos de capturar dichos eventos. En cualquier caso se trata de una tarea compleja y que requiere una gran comprensión de la naturaleza de los comportamientos de tráfico de red.

Capítulo 9.

9. DISEÑO E IMPLEMENTACIÓN DE LA APLICACIÓN

9.1 INTRODUCCIÓN

En esta sección se aborda el diseño y la estructura que hemos seguido y utilizado para desarrollar la aplicación de un modo coherente y modulable. Se especifican ciertos detalles de implementación con un alto nivel de abstracción de modo divulgativo y para facilitar una rápida comprensión de la aplicación.

Nuestro objetivo ha sido crear una aplicación que fuera modular y extensible por lo que hemos hecho uso de patrones de programación orientada a objetos para dicho fin.

El código de la aplicación está dividido en cuatro paquetes *Java* denominados: *Vista, Controlador, Modelo* y *Main*.

9.2 DISEÑO DE LA APLICACIÓN

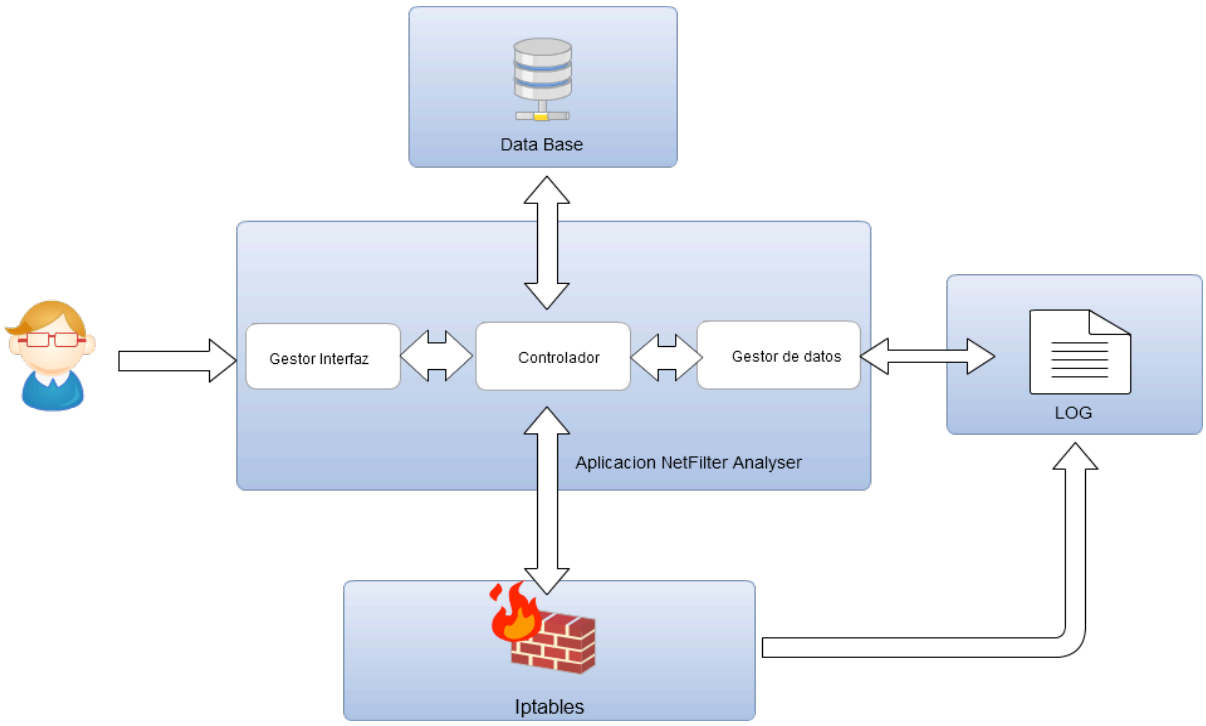


Figura 33: Diseño de la aplicación

9.2.1 MÓDULOS DE LA APLICACIÓN

La aplicación está diseñada en módulos. Algunos de ellos nos proporcionan funcionalidades y otros simplemente se encargan de gestionar a otros módulos. A continuación, destacamos los más importantes y detallamos sus características principales.

9.2.1.1 Gestor de datos

Este es el módulo principal de la aplicación. Se encarga de gestionar todo el ámbito de los datos. Dicho módulo provee al diseño de la aplicación de un grado elevado de abstracción. Tal es así que no supondría un gran esfuerzo el cambiar de algún modo el modelo de datos sin repercutir en el rendimiento de la aplicación.

La concordancia de dicho módulo en el código fuente de la aplicación la encontramos en el paquete denominado *modelo* y en particular en la clase llamada *GestorModelo*.

Dicho módulo gestiona los siguientes módulos:

9.2.1.1.1 Gestor de reglas

Dicho gestor nos proporciona la funcionalidad de generar las reglas de acuerdo con los parámetros introducidos por el interfaz. De este modo, el generador de reglas en primer lugar crea un objeto denominado *Datos Iniciales* en el que se guardan los parámetros que comprende cualquier regla y son generales a todas ellas.

Tales parámetros son el identificador de regla, la cadena en la que se va a insertar y el protocolo (TCP, UDP, ICMP). Todos estos parámetros son obligatorios y no se podrá generar una regla sin que se le haya dado valor a alguno de ellos.

Además de ello también guardará en dicho objeto la cadena a buscar en caso de que el usuario hubiese especificado alguna. Tal y como se sobreentiende dicho parámetro es opcional en la generación de la regla.

El resto de parámetros dependen del protocolo escogido por el usuario y se guardarán en un objeto generado por las clases correspondientes: *DatosUDP, DatosTCP* o *DatosICMP*.

Una vez que dicho gestor genera dichos objetos, crea en último lugar el objeto regla en el que se parametrizan todos estos datos y en el que se guarda en otro atributo la regla con la sintaxis de *Iptables* formulada.

Dicho objeto es generado por la clase *ReglaIptables*. Tal y como hemos especificado anteriormente todo este volumen de datos se encuentra encuadrado dentro del paquete *Modelo*.

Gracias a esta gestión, insertar nuevas opciones a la generación de reglas supone una tarea sencilla y no plantearía grandes dificultades.

9.2.1.1.2 Gestor de biblioteca

A fin de dar versatilidad y profundidad a la aplicación decidimos incorporar la posibilidad de guardar las reglas en distintas bibliotecas.

El sistema genera una biblioteca por defecto llamada *Sistema*. Dicho gestor nos permite realizar inserciones, búsqueda y borrado de reglas de forma eficiente y segura.

Este gestor trabaja con la clase *Biblioteca* que tiene como identificador el atributo *nombreBiblioteca* y a su vez un vector de objetos *ReglaIptables*.

9.2.1.1.3 Gestor de persistencia

Este módulo se encarga de insertar las reglas que hayamos generado con la ayuda de los formularios en la base de datos.

Dicho gestor se encuentra en la clase denominada *Persistencia* y guarda instancias del *gestorBiblioteca* y del *lectorLog*. Por supuesto, dicho gestor también es invocado opcionalmente cuando se arranca el programa para que cargue en la memoria las reglas guardadas en la base de datos.

Las operaciones básicas son realizadas por los métodos *escribirEnFichero* y *leerDeFichero*.

9.2.1.1.4 Gestor del Log

Este módulo se encarga de analizar el *Log* adecuadamente y de buscar si dicho archivo contiene información que nos informe de algún comportamiento relevante. Para realizar dichas funciones el gestor del *Log* hace uso de las siguientes clases *Java*: *LectorLog*, *entity* y *Log*.

El resultado de analizar el *Log* se muestra de forma visual con lo que este gestor tiene la responsabilidad de informar al controlador de cualquier evento que requiera un cambio en el visor del *Log*.

9.2.1.2 Controlador

Este módulo se encuentra dentro del paquete *Java* denominado *Controlador* y a su vez dentro de la clase *Java* con nombre *ControladorPrincipal*.

Dicho módulo tiene sentido dentro del patrón *Modelo Vista Controlador* que hemos seguido para el desarrollo de la práctica y del que se hace especial énfasis en el apartado de detalles de la implementación.

Dicho controlador guarda en sus atributos instancias a las diferentes tipos de vistas y al gestor del modelo. Por lo tanto supone un elemento central entre la vista y el controlador capaz de notificar cualquier cambio a dichas entidades de forma eficiente y localizada.

9.2.1.3 Gestor de Interfaces

Este gestor de interfaces se encuentra en el paquete *Java* denominado *Interfaz* y es capaz de gestionar todos los posibles interfaces o vistas que se pueden generar durante una ejecución cualquiera de nuestra aplicación.

Dicho gestor recibe las órdenes provenientes del controlador que notifican de algún cambio en el modelo y de la consiguiente necesidad de reconfigurar su representación visual. Las clases más relevantes con las que trabaja dicho módulo son las siguientes: *VistaPrincipal*, *InterfazTCP*, *InterfazUDP*, *InterfazICMP*, *VisorLog*.

Dichas interfaces las hemos realizado con ayuda de la biblioteca *Swing*. Esta biblioteca incluye *widgets* para la interfaz gráfica del usuario tales como cajas de texto, botones, desplegables y tablas. Sigue un simple modelo de programación por hilos y posee las siguientes características principales:

- Independencia de plataforma.
- Modular.
- Extensible.
- Personalizable.

9.2.1.4 Interconexión de los módulos

Tal y como hemos especificado previamente, los módulos están conectados de forma muy simple gracias al controlador. No hay conexión directa entre los datos y la vista, de forma que cualquier modificación en una de esas entidades no supone ninguna refactorización en su otra representación.

9.3 DIAGRAMA DE CONTROL DE FLUJO DE LA APLICACIÓN

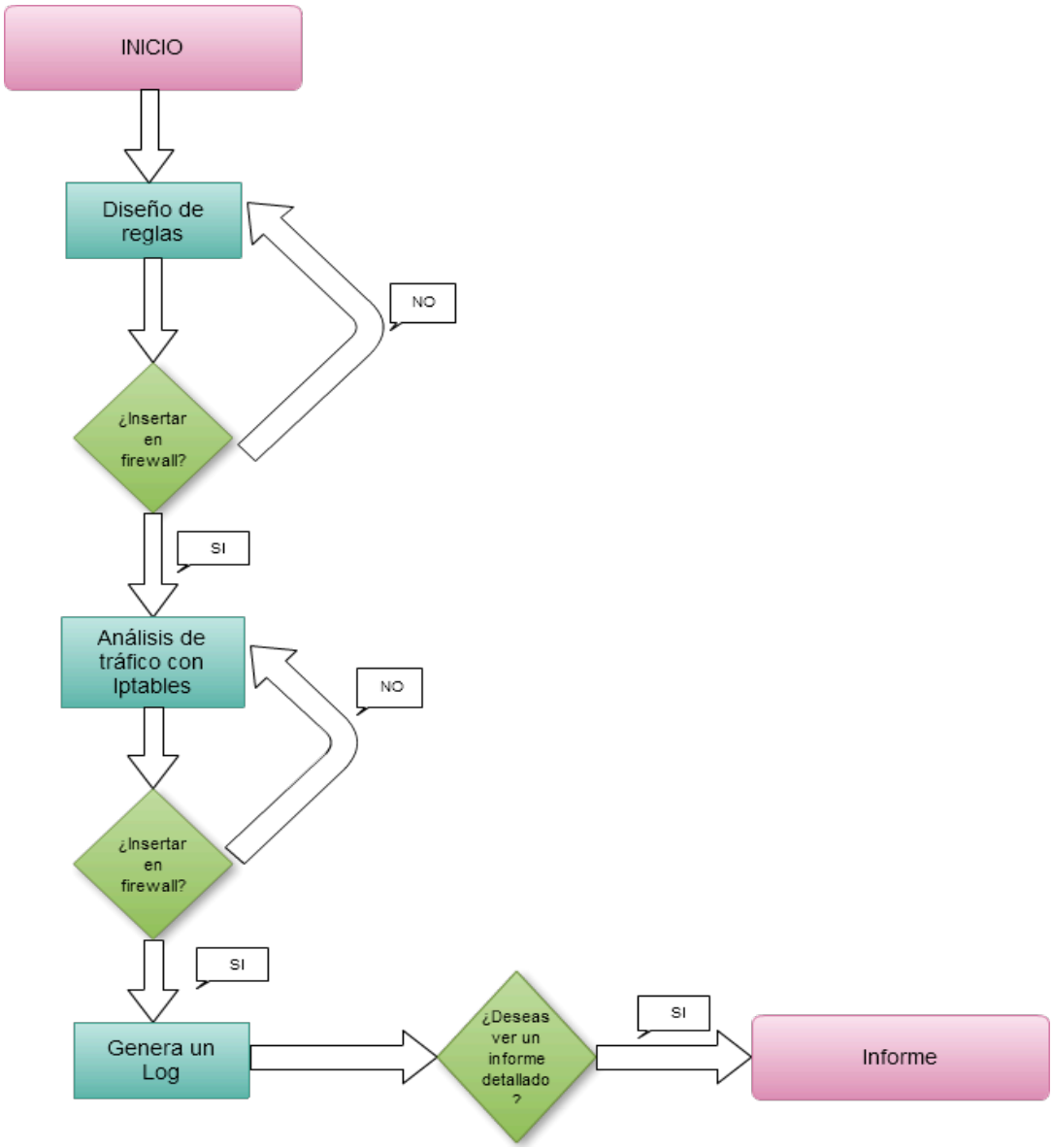


Figura 34: Diagrama de control de flujo

9.4 DETALLES DE LA IMPLEMENTACIÓN

En primer lugar cabe destacar que la implementación ha sido realizada en el lenguaje de programación *Java*. Esto es debido a diversas razones entre las que podemos encontrar la pericia de los programadores en este lenguaje y la potencia y versatilidad de este lenguaje.

Uno de los aspectos más importantes a tener en cuenta en la elección de dicho lenguaje fue que nos ha permitido probar la aplicación en múltiples sistemas operativos gracias a que la tecnología *Java* es multiplataforma. Por lo tanto, a pesar de que la aplicación sólo tiene sentido en un entorno *GNU/*Linux ésta puede ser ejecutada tanto en máquinas *Macintosh* como máquinas con sistema operativo *Microsoft Windows*.

Por otra parte el lenguaje en sí mismo toma mucha sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria.

A continuación, se describe el patrón de arquitectura de *software* seguido para el desarrollo de la aplicación.

Modelo Vista Controlador:

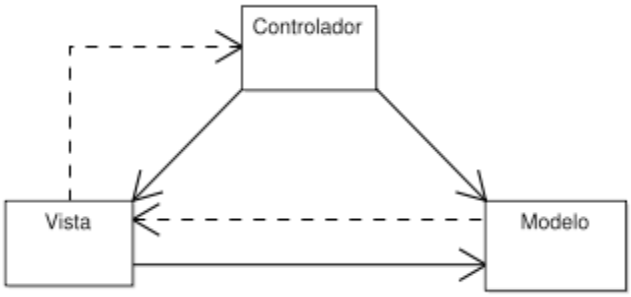


Figura 35: Arquitectura Modelo-Vista-Controlador

El modelo es el responsable de:

- Acceder a la capa de almacenamiento de datos. Lo ideal es que el modelo sea independiente del sistema de almacenamiento.
- Definir la funcionalidad del sistema.
- Llevar un registro de las vistas y controladores del sistema.
- Notificar a las vistas los cambios de los datos que pueda producir un agente externo.

El controlador es responsable de:

- Recibir los eventos de entrada.
- Mantener las reglas de gestión de eventos tales como solicitar peticiones al modelo o a las vistas.

Las vistas son responsables de:

- Recibir datos del modelo y mostrarlos al usuario.
- Tener un registro de su controlador asociado.

Capítulo 10.

10. MANUAL DE USUARIO

10.1 INTRODUCCIÓN

Este manual de usuario ha sido elaborado con la intención de ofrecer la información necesaria para el uso de la aplicación.

Un usuario sin experiencia en el manejo de la aplicación *Netfilter Analyser* debería ser capaz de realizar cualquier función con la lectura de este capítulo.

10.2 OBJETIVO DE LA APLICACIÓN

La aplicación *NetFilter Analyser* tiene por objetivo ser capaz de alertar en tiempo real de eventos en la Red. Dichos eventos son caracterizados por el usuario gracias a las reglas de iptables que generará por medio de la aplicación.

La aplicación por tanto tiene como objetivo primordial la generación de reglas tanto de forma cómoda como experta y a su vez, la lectura de los resultados de dichas reglas y la notificación al usuario en caso de dichos eventos en la Red.

10.3 REQUISITOS DEL SISTEMA

Netfilter Analyser es *software* que se ejecuta en *Windows*, *Mac* y *Linux*. Sin embargo, debido a que *Iptables* corre en *Linux*, la aplicación sólo podrá realizar toda su completa actividad en una plataforma de *Linux*. No obstante, el *Log* siempre se podría analizar de manera remota.

Requisitos del sistema *Linux*

Plataforma	Versión	Memoria	Navegadores	Espacio en disco
Linux (32 bits)				
Linux x86				
	Oracle Enterprise Linux 5.5, 5.4	64 MB	, Firefox 3.0.x, Firefox 3.5.x o Firefox 3.6	58 MB
	Red Hat Enterprise Linux 5.3, 5.4,	64 MB	, Firefox 3.0.x, Firefox 3.5.x o Firefox 3.6	

	5.5			
	Red Hat Enterprise Linux AS/ES 4.0 (4.1-4.8)	64 MB	Firefox 2, Mozilla 1.4 Mozilla 1.7	
	SUSE 10	64 MB	Firefox 2.0.x	
	SUSE 9	64 MB	Mozilla 1.4	
	SLES 11	64 MB	Firefox 3.0.6, Firefox 3.0.8	
Linux (64 bits)				
Linux x64 Modo de 64 bits	Oracle Enterprise Linux 5.5, 5.4	64 MB	**Sistemas operativos de 64 bits, navegadores de 32 bits:** Firefox 3.0.x, Firefox 3.5.x, o Firefox 3.6	58 MB
	Red Hat Enterprise Linux 5.3, 5.4, 5.5	64 MB	**SO de 64 bits, navegadores de 32 bits:** , Firefox 3.0.x, Firefox 3.5.x o Firefox 3.6	
	Red Hat Enterprise Linux 5.0 (5.1, 5.2)	64 MB	**SO de 64 bits, navegadores de 32 bits:** Firefox 3.0.x, Firefox 3.0.x,	
	Red Hat Enterprise Linux AS/ES 4.0 (4.1-4.8)	64 MB	**SO de 64 bits, navegadores de 32 bits:** Firefox 2.0.x Mozilla 1.4.X o 1.7,	
	SUSE 8.2, 9, 9.1, 9.2	64 MB	**SO de 64 bits, navegadores de 32 bits:** Mozilla 1.4	
	SLES 11	64 MB	**SO de 64 bits, navegadores de 32 bits:** Firefox 3.0.6 o 3.0.8	
	SLES 10	64 MB	**SO de 64 bits, navegadores de 32 bits:** Firefox 2.0.x	
	SLES 9, 8	64 MB	**SO de 64 bits, navegadores de 32 bits:** Mozilla 1.4	

Figura 36: Requisitos del sistema Linux

10.4 FUNCIONALIDADES OBLIGATORIAS

Las funcionalidades que deberá aportar el producto desarrollado son:

- Generar reglas con la sintaxis de *Iptables*.
- Gestionar (añadir, mostrar, eliminar, guardar) las reglas en el *firewall*.
- Gestionar (añadir, agrupar, mostrar, eliminar, guardar) las reglas en colecciones de reglas.
- Gestionar (crear, mostrar, eliminar, insertar) las colecciones de reglas.
- Analizar el fichero "*Log*".
- Abrir y mostrar el contenido de un fichero "*Log*".
- Alertar de reglas detectadas en el "*Log*".

10.5 ESCENARIOS DE CALIDAD

Fiabilidad. Se debe garantizar el correcto almacenamiento de los datos del usuario y su persistencia. Además, se deberán proporcionar mecanismos de copia de seguridad para facilitar la recuperación en caso de fallo.

Portabilidad. Dado que el sistema será una aplicación *Java* estándar, podrá funcionar en todas las plataformas en que esté disponible la versión 1.5 ó 1.6 de *JVM*.

Seguridad. El sistema deberá contar con las medidas técnicas necesarias para garantizar la seguridad de datos que circulan por la Red.

Sostenibilidad. El sistema estará dividido en módulos, lo que facilitará el mantenimiento del *software*. Además, se adjuntará la documentación necesaria sobre la aplicación y el modo en que está construida. De esta forma, las personas encargadas del mantenimiento de la aplicación podrán contar con material de apoyo para familiarizarse con el sistema.

Disponibilidad. La disponibilidad de nuestro sistema estará siempre garantizada. La aplicación deberá ser lo más robusta posible para garantizar las alertas en tiempo real.

Versatilidad. El sistema debe permitir conmutar con rapidez con *Iptables* con el fin de garantizar la eficiencia de la aplicación.

10.6 INSTALACIÓN DE LA APLICACIÓN

Para ejecutar la aplicación es necesario tener instalados previamente dos aplicaciones: *Java* e *Iptables*. A continuación, se expone una explicación de los pasos a seguir para su correcta instalación en *UNIX*.

Prerrequisitos: Antes de empezar a instalar las aplicaciones es necesario asegurarse de que el ordenador está conectado a Internet.

10.6.1 INSTALACIÓN JAVA

Desde los repositorios de *Ubuntu*

Por defecto, en *Ubuntu* disponemos de una versión libre del JRE (*Java Runtime Environment*, Entorno de Ejecución de Java) de *Java*. Actualmente, *Java* ya se encuentra en la versión 6 de su desarrollo.

Para instalar el JRE6 o el JDK6 simplemente debemos instalar desde los repositorios los paquetes *sun-java6-bin, sun-java6-jre* y *sun-java6-jdk*, respectivamente.

Para ello introducimos el siguiente comando:

sudo apt-get install sun-java6-bin, sun-java6-jre, sun-java6-jdk

Desde la web de *Java*

Otra opción válida es descargarse de la página web de *Sun* el JRE6 o, en caso de que se desee programar, el JDK6. Una vez descargado, tenemos que cambiarle los permisos al fichero para que se pueda ejecutar e instalarlo. Se puede usar el tutorial de instalación de *Java* disponible en la siguiente URL: *http://www.java.com/es/download/help/linux_install.xml*

10.6.2 INSTALACIÓN IPTABLES

Se instala *Iptables* ejecutando el siguiente comando:

sudo apt-get install iptables

10.7 EJECUTAR LA APLICACIÓN

Para ejecutar la aplicación basta con ejecutar el archivo con extensión *jar*. Un archivo JAR (*Java ARchive*) es un tipo de archivo que permite ejecutar aplicaciones escritas en lenguaje *Java*.

Dicha aplicación requiere permisos de *root* o superusuario. Por lo tanto si el usuario desea poder disfrutar de todas las funcionalidades que brinda la aplicación *Netfilter Analyser* deberá ejecutar la aplicación con el siguiente comando:

sudo java –jar ProyectoFinCarrera.jar

```
manumarugan@ubuntu:~/Desktop/Proyecto/Version 25 Abril/ProyectoFinCarrera/dist$ sudo java -jar ProyectoFinCarrera.jar
```

Figura 37: Comando para ejecutar la aplicación

10.7.1 CREAR UNA REGLA

La aplicación, entre otras funcionalidades, posee la de generar reglas de *Iptables* e insertarlas en el *kernel*. Para crear las reglas, la aplicación nos proporciona dos posibilidades: crear la regla en modo principiante o en modo experto.

10.7.1.1 Crear una regla modo usuario principiante

Para crear un regla en modo principiante el usuario deberá seleccionar en el menú la opción "*Add new rule simply"*. Dicha opción se encuentra en el submenú "*Rules->Add rules".*

A continuación, se abrirá la siguiente ventana:

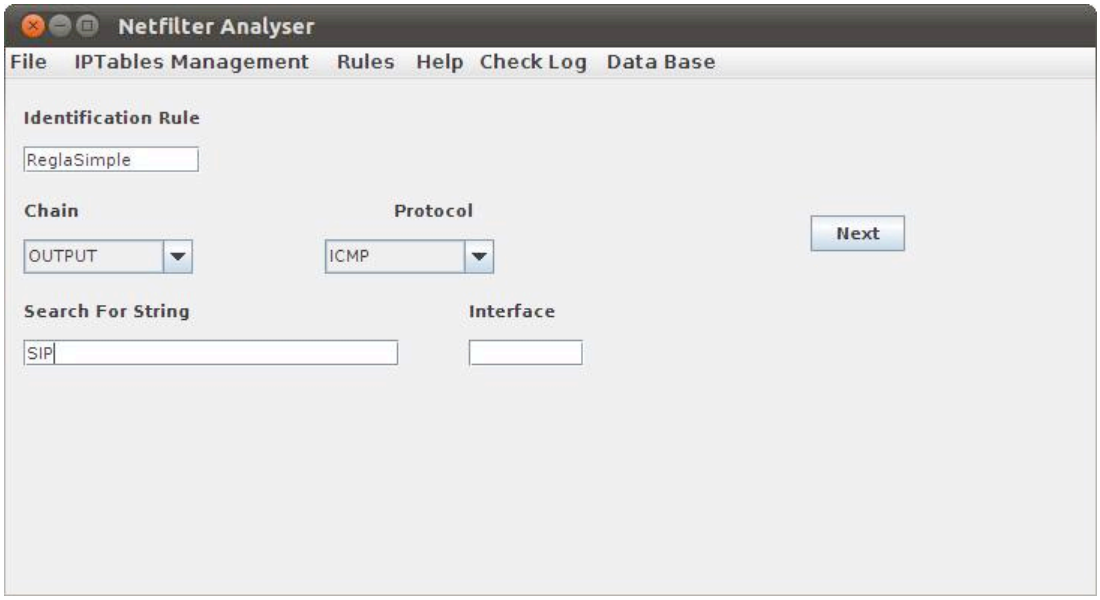


Figura 38: Ventana para crear regla en modo usuario principiante

La interfaz está compuesta por una serie de campos de texto a rellenar. De entre ellos los campos identificador regla, protocolo y cadena son obligatorios. En caso de no dar valor a alguno de estos campos la aplicación nos lo advertirá y no nos permitirá continuar al siguiente punto.

Una vez que hayamos insertado los datos iniciales y pulsado el botón "*Next",* la aplicación nos pregunta en qué biblioteca queremos insertar la regla. Por defecto, podremos pulsar la biblioteca "*Sistema"*, pero si el usuario ya ha creado alguna biblioteca previamente, ésta, también aparecerá disponible.

A continuación, el usuario podrá rellenar los datos avanzados pertenecientes a la regla en función del protocolo que escogió. Ninguno de los siguientes datos serán obligatorios pudiendo dejar todo vacío y pulsar directamente el botón "*Add"*.

Las interfaces que se muestran en relación al protocolo seleccionado son las siguientes:

Protocolo ICMP:

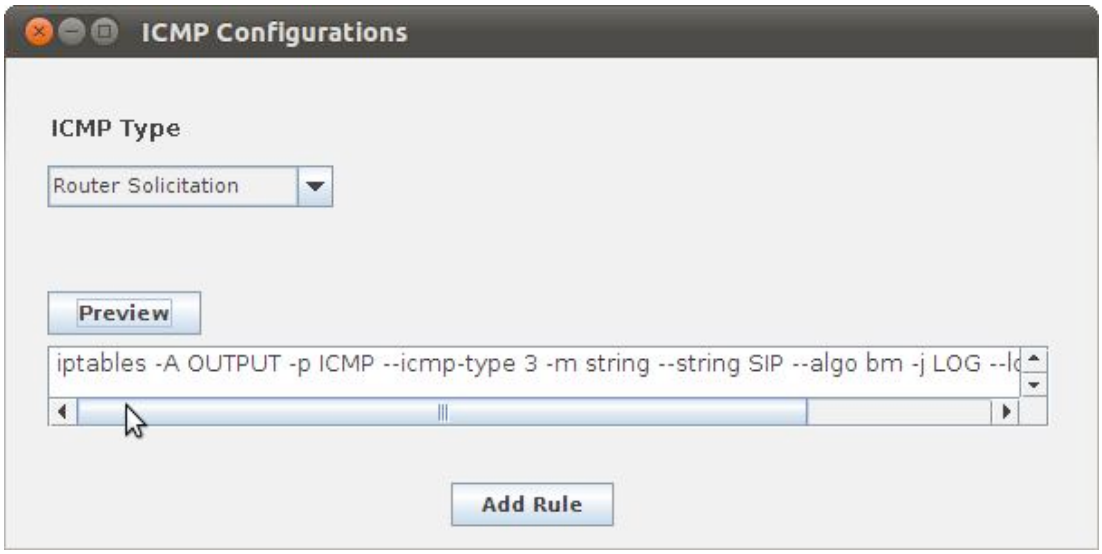


Figura 39: Crear regla modo principiante protocolo ICMP

Protocolo TCP:

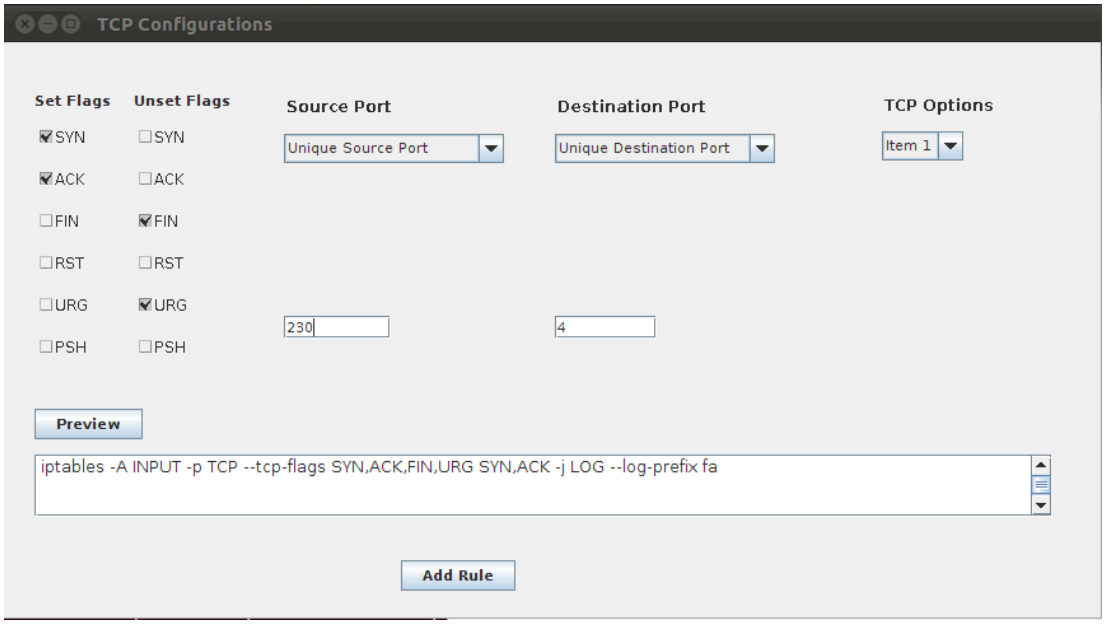


Figura 40: Crear regla modo principiante protocolo TCP

Protocolo UDP:

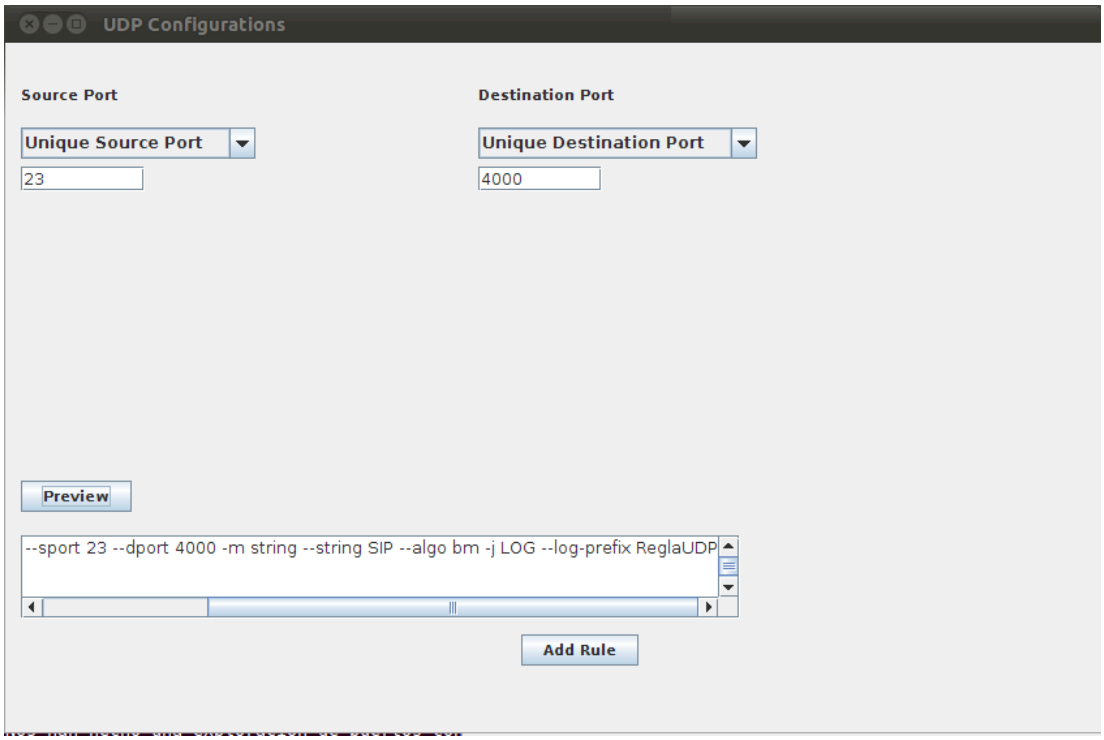


Figura 41: Crear regla modo principiante protocolo UDP

Por último, la aplicación nos preguntará si estamos seguros de crear la regla, si queremos insertarla en el *firewall* o dejar su inserción para más tarde. La ventana asociada a dicha acción es la siguiente:

Figura 42: Mensaje de aviso de inserción de reglas en Iptables

10.7.1.2 Crear una regla modo usuario avanzado

La aplicación nos permite insertar reglas de forma experta. Esta opción está destinada a un usuario que ya está familiarizado con la sintaxis de *Iptables* y no desea tener que rellenar ningún formulario. Por defecto, la aplicación insertará las reglas que el usuario haya generado.

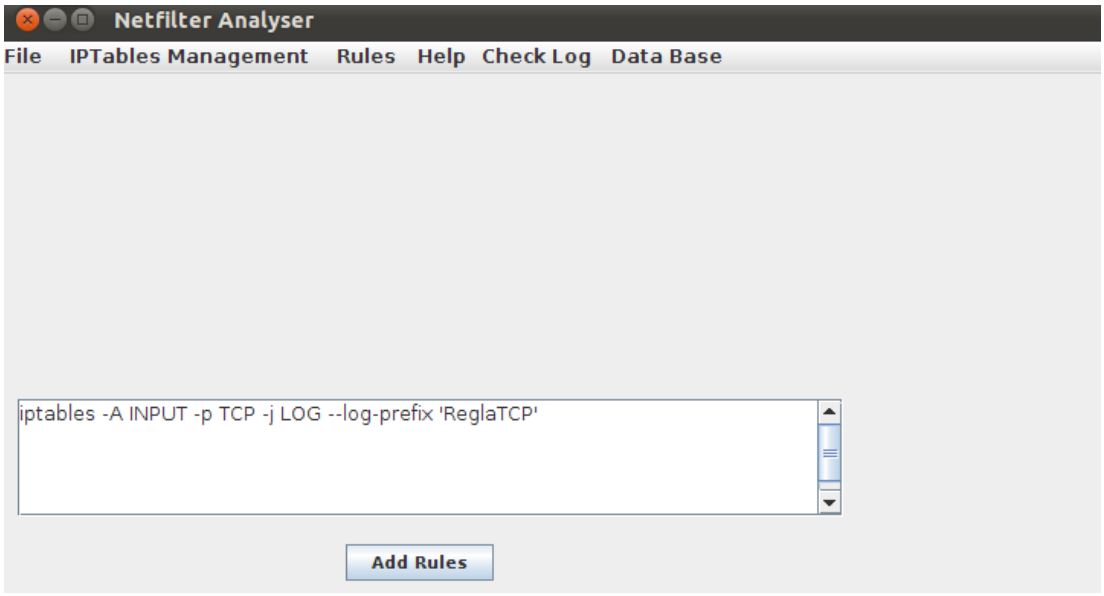


Figura 43: Crear regla en modo usuario avanzado

10.7.2 CREAR UNA COLECCIÓN DE REGLAS

La aplicación nos permite generar colecciones de reglas distintas de la que el sistema nos proporciona por defecto. Dicha funcionalidad nos es de gran interés ya que, una vez que tenemos creada una colección, podemos asignar cualquier conjunto de reglas a dicha colección.

Para crear una nueva colección basta con pulsar el menú *Rules* y a continuación, seleccionar el botón "*Create a new collection of rules"*.

Figura 44: Ventana para insertar nombre colección

10.7.3 LISTAR REGLAS CREADAS

Esta opción nos permite visualizar un listado de las reglas que hayan sido generadas con la aplicación. En dicho listado podremos ver los parámetros más importantes, tales como su identificador, el protocolo que usa y la biblioteca a la que pertenece.

Para activar dicha opción basta con seleccionar el botón del menú principal: "*Show actual rules"*.

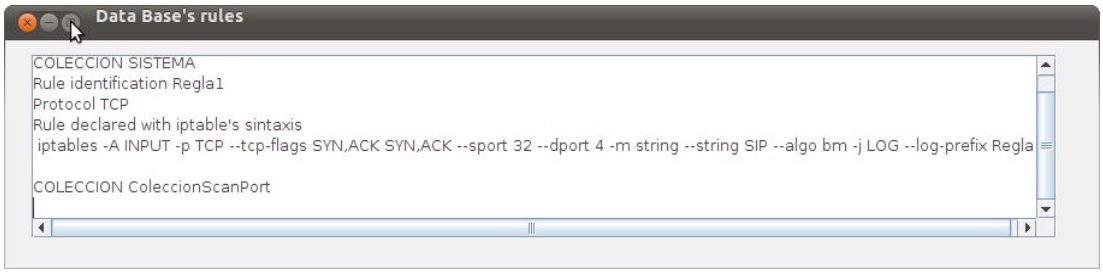


Figura 45: Listado reglas de la base de datos

10.7.4 BUSCAR UNA REGLA

Esta opción permite al usuario realizar la búsqueda de una regla conociendo su identificador. Para ello, basta con pulsar el botón "*Rules"* y a continuación, seleccionar el botón *"Search for rules"* del menú principal.

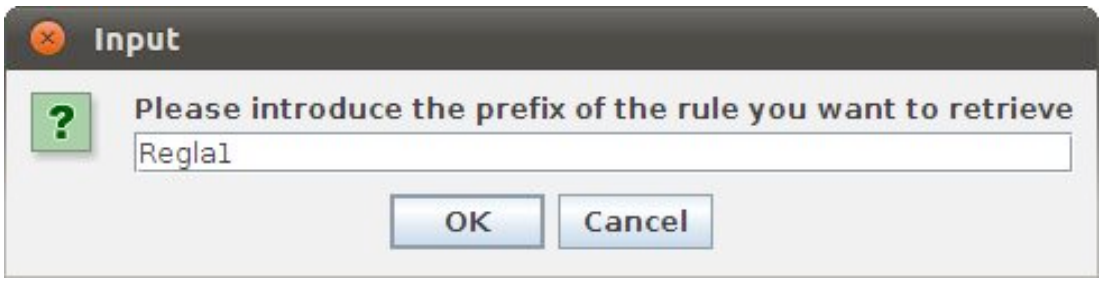


Figura 46: Introducción del nombre de la regla a buscar

La siguiente ventana es la que se muestra como resultado de buscar una regla.

Figura 47: Ventana que contiene la regla buscada

10.7.5 BORRAR UN REGLA

Esta opción permite al usuario quitar una regla de la base de datos del sistema. Dicha regla no será cargada de nuevo en la base de datos una vez que se abandone la ejecución actual de la aplicación. Para ello, basta con seleccionar el botón "*Delete Rules"* y posteriormente seleccionar el botón correspondiente a *"Delete a certain rule"* del menú principal.

A continuación, la aplicación nos preguntará por el identificador de la regla. En caso de no conocerlo, podemos buscarlo en la opción correspondiente a *Listar Reglas Creadas* descrita previamente.

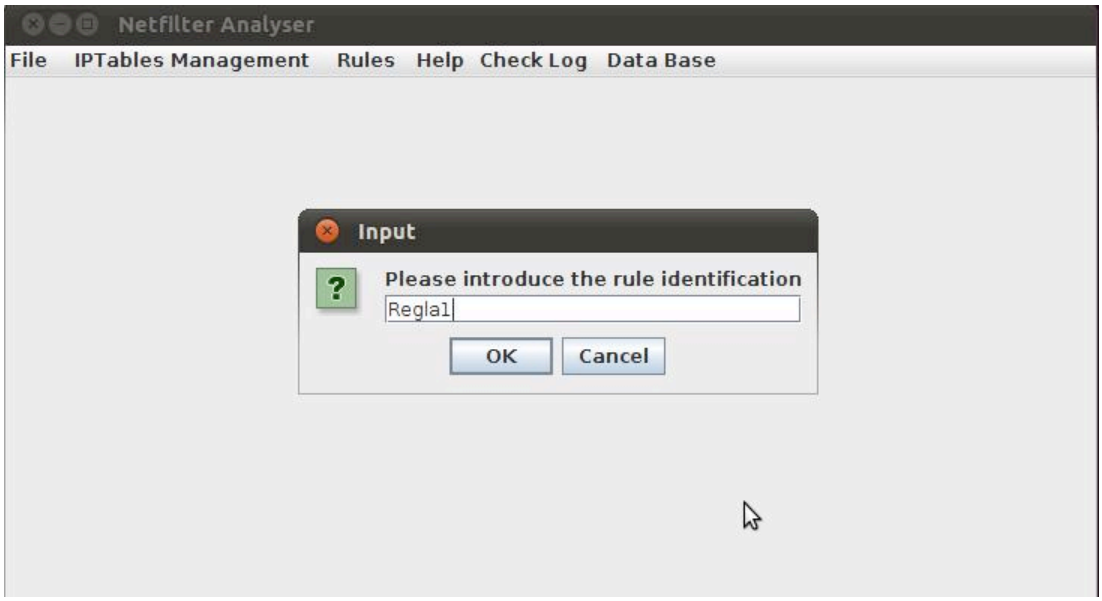


Figura 48: Ventana para introducir el nombre de la regla a eliminar

Por último la aplicación muestra un mensaje informando que la regla ha sido eliminada correctamente.

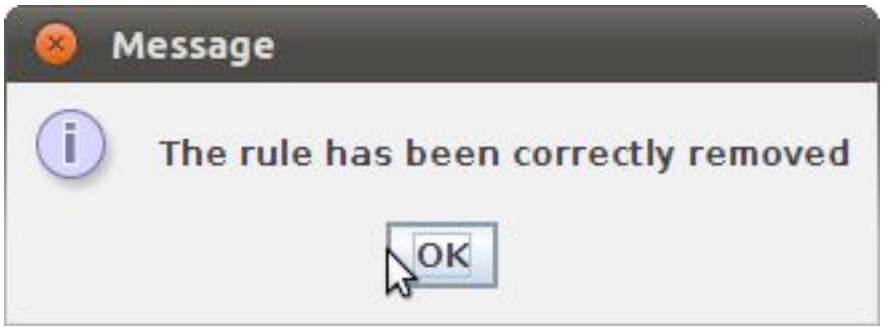


Figura 49: Mensaje informativo de regla eliminada correctamente

10.7.6 BORRAR TODAS LAS REGLAS

Para borrar todas las reglas el usuario deberá seleccionar la opción del menú "*Iptables Management"* y a continuación pulsar sobre el botón "*Delete all Rules"*.

La aplicación mostrará un mensaje indicando si dicha acción ha sido efectuada con éxito o no. En caso de que no hubiera ninguna regla insertada de antemano, la aplicación muestra un mensaje tal y como se puede apreciar en la figura posterior.

Figura 50: Mensaje informativo de reglas ya borradas

10.7.7 BORRAR UNA COLECCIÓN DE REGLAS

Esta opción es similar a la descrita anteriormente con la salvedad de que en este caso lo que eliminaremos será una colección entera de reglas. En este caso, la aplicación nos preguntará por el nombre de la biblioteca. En caso de no conocerla podemos listar las reglas creadas y obtener los identificadores de las bibliotecas.

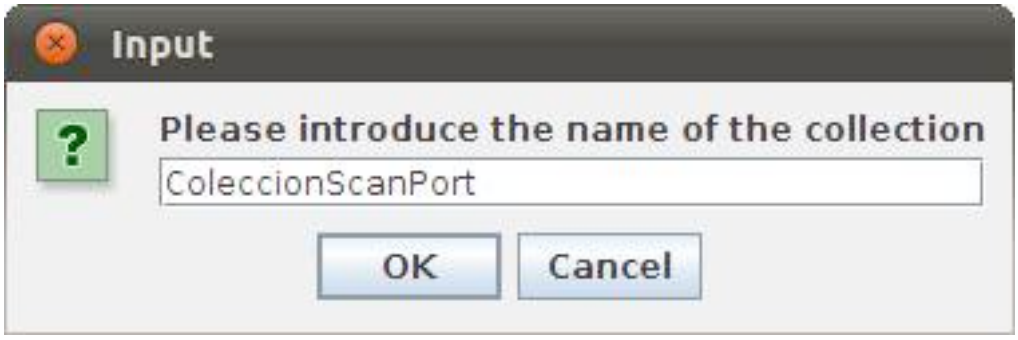


Figura 51: Introducción nombre de colección de reglas a borrar

Por último la aplicación muestra un mensaje informando que la colección de reglas ha sido eliminada correctamente.

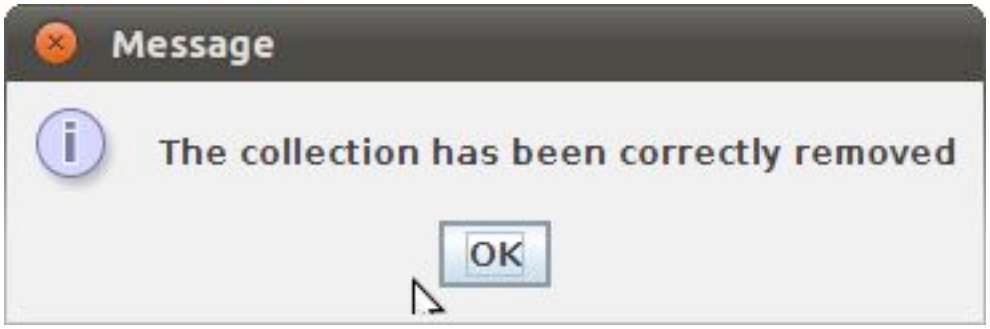


Figura 52: Mensaje informativo colección eliminada correctamente

10.7.8 INSERTAR UN REGLA POR ID

Esta opción nos permite insertar una regla en el *firewall* que haya sido generada con anterioridad y que aún no haya sido insertada. Recordemos que al generar una regla podemos dejar para más tarde su inserción.

En este caso basta con pulsar el botón del menú principal: "*Iptables Management"* y a continuación seleccionar el botón "*Insert rule by rule ID"*.

A continuación, la aplicación nos preguntará por el identificador de la regla. Una vez que lo hayamos escrito y confirmado que queremos insertar la regla, la aplicación mostrará al usuario un mensaje informando de si dicha acción ha sido realizada con éxito o no.

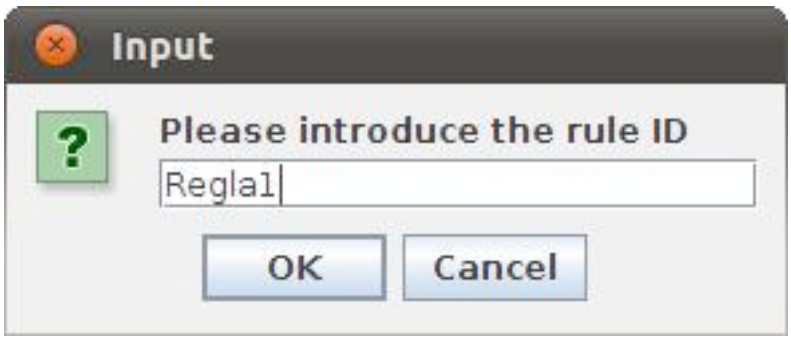


Figura 53: Ventana para introducir el nombre de la regla a insertar

Por último la aplicación muestra un mensaje informando que la regla ha sido insertada correctamente en *Iptables*.

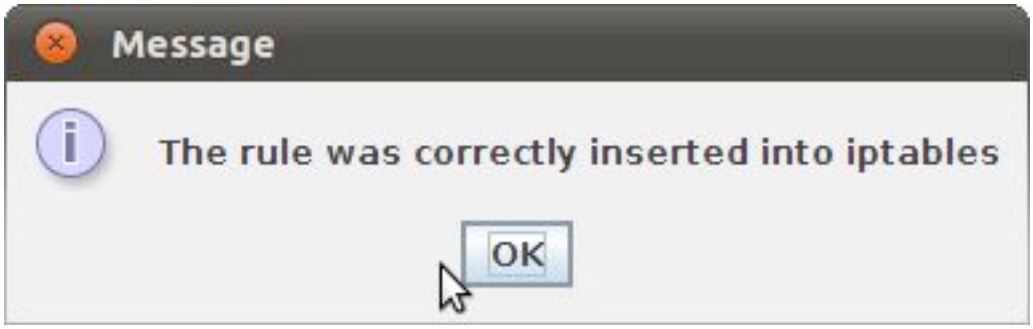


Figura 54: Mensaje informativo de regla insertada correctamente

10.7.9 INSERTAR UNA COLECCIÓN

Esta opción es similar a la descrita con anterioridad con la salvedad de que en este caso permite al usuario insertar una colección de reglas completa. Dicha opción resulta de gran utilidad puesto que una colección puede contener numerosas reglas e insertarlas una a una podría resultar tedioso.

Para realizar dicha acción basta con pulsar en "*Insert rules of a collection"* y escribir el nombre de la colección. La aplicación informará al usuario acerca del éxito de dicha operación.

10.7.10 INSERTAR REGLAS DE PATRONES YA CONFIGURADAS.

Esta opción permite al usuario insertar reglas que ya han sido creadas y que alertarán al usuario en caso de que se cumplan las condiciones de dichas reglas. De entre las opciones ya configuradas podemos encontrar: insertar reglas para detectar una exploración de puertos, insertar reglas para detectar un ataque de denegación de servicio o insertar reglas para detectar un uso de VoIP.

Dichas reglas ya han sido creadas y se encuentran en el repositorio de la aplicación. El usuario sólo tendrá que activarlas para que sea alertado en caso de que se dé alguno de esos comportamientos en el tráfico de la Red.

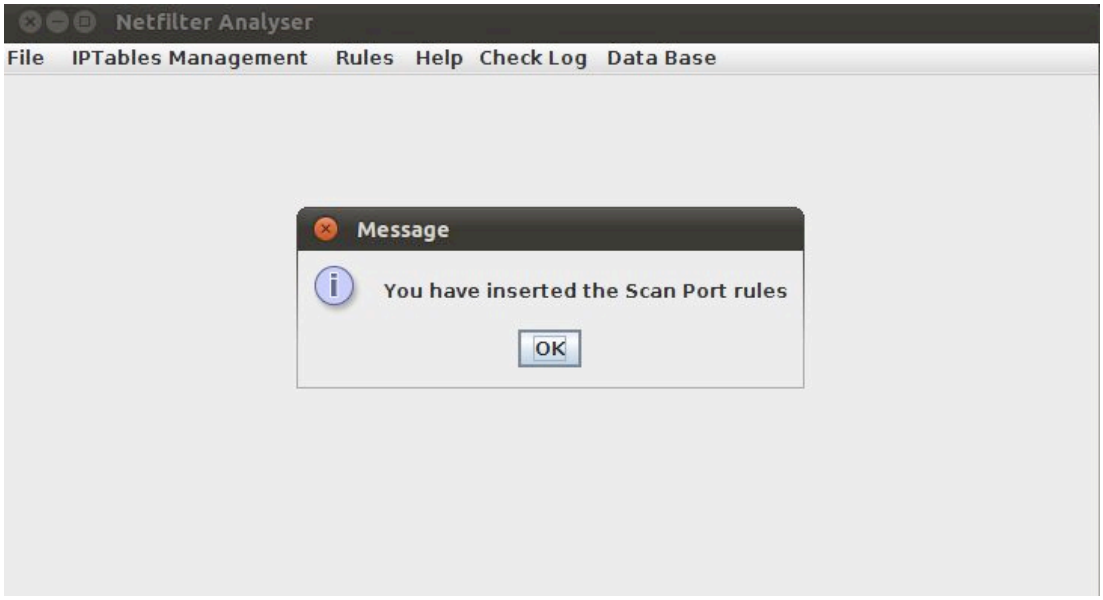


Figura 55: Mensaje patrón "exploración de puertos" insertado

10.7.11 MOSTRAR LAS REGLAS DE IPTABLES

La opción para ver las reglas que están insertadas en *Iptables* es "*Rules"* y a continuación seleccionar el botón *"Show Rules"*.

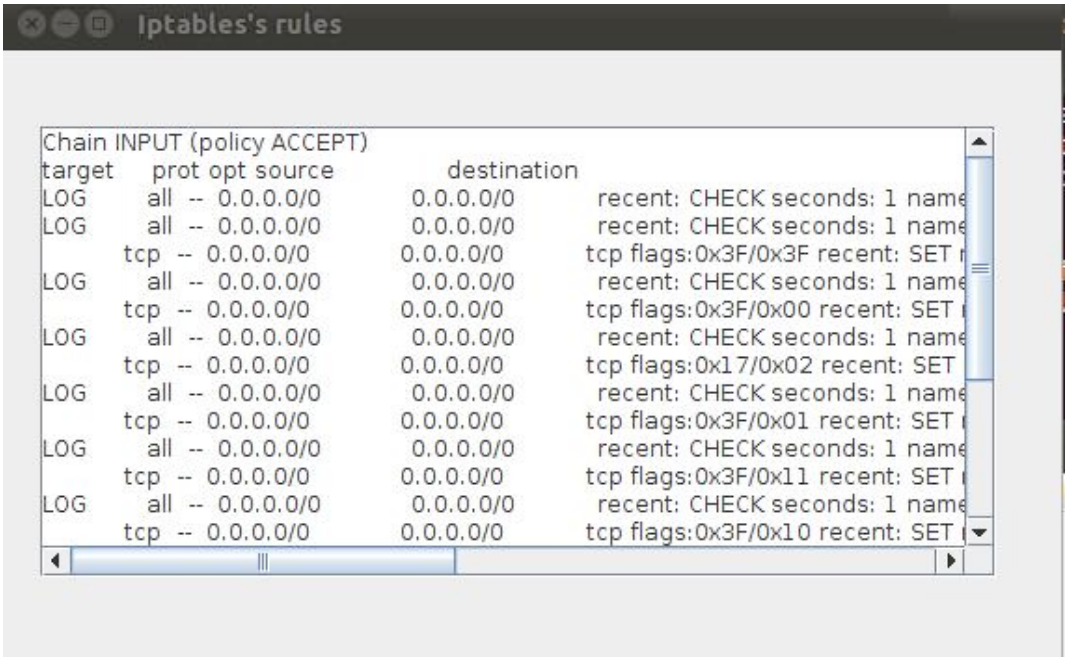


Figura 56: Ventana con las reglas que contiene *Iptables*

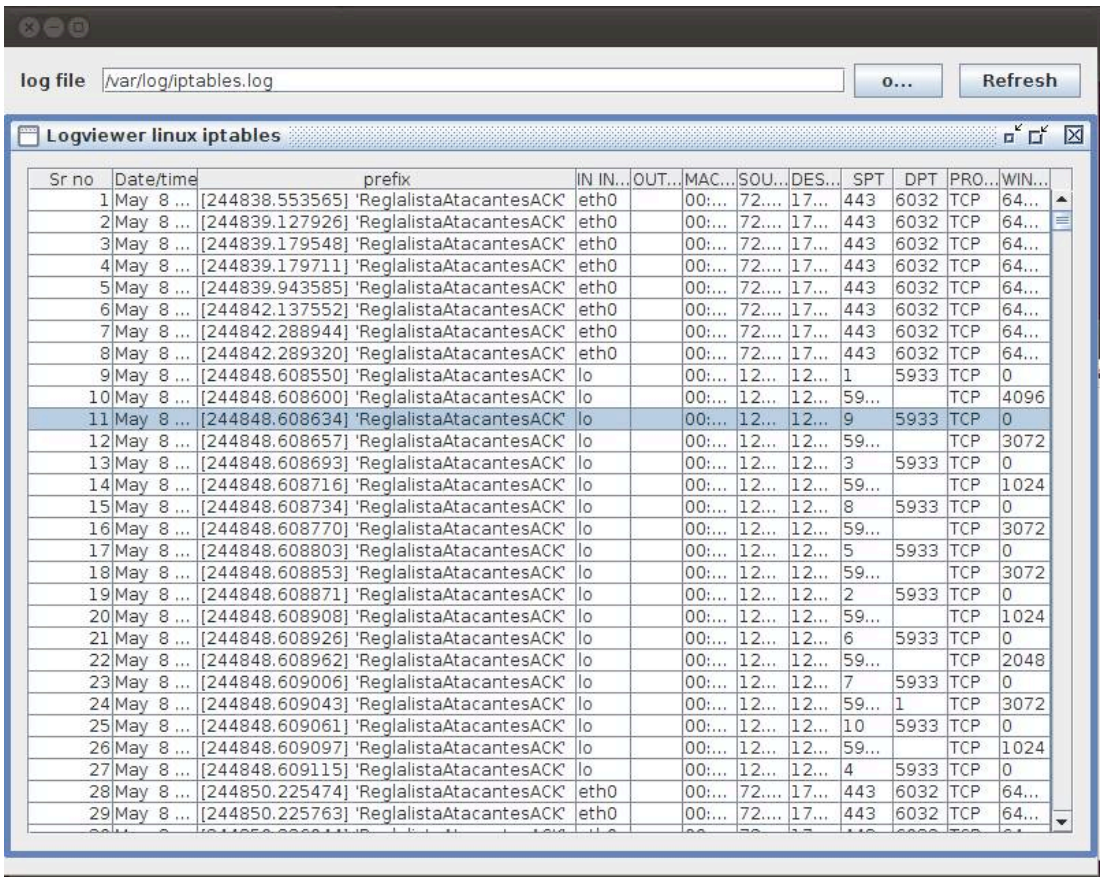
10.7.12 MOSTRAR UN ARCHIVO LOG

Esta opción permite al usuario mostrar un archivo *Log* cualquiera de *Iptables*. Para ello, el usuario deberá seleccionar la opción "*Check Log"* y a continuación seleccionar el botón *"Show Log"*.

Acto seguido, el usuario deberá navegar a través de las carpetas del sistema hasta encontrar el *Log* que le interese visualizar.

La aplicación se encargará de analizar el *Log* y mostrarlo de una forma cómoda de leer para el usuario. Además de ello, en caso de que se haya producido tráfico que el usuario haya caracterizado por medio de las reglas, la aplicación alertará de ello y mostrará un breve informe de dicho tráfico.

En dicho informe podremos ver la dirección IP origen, el prefijo que el usuario puso en la regla y la fecha y hora en la que se produjo.



Sr no	Date/time	prefix	IN IN...	OUT...	MAC...	SOU...	DES...	SPT	DPT	PRO...	WIN...
1	May 8 ...	[244838.553565] 'ReglalistaAtacantesACK'	eth0		00:...	72....	17...	443	6032	TCP	64...
2	May 8 ...	[244839.127926] 'ReglalistaAtacantesACK'	eth0		00:...	72....	17...	443	6032	TCP	64...
3	May 8 ...	[244839.179548] 'ReglalistaAtacantesACK'	eth0		00:...	72....	17...	443	6032	TCP	64...
4	May 8 ...	[244839.179711] 'ReglalistaAtacantesACK'	eth0		00:...	72....	17...	443	6032	TCP	64...
5	May 8 ...	[244839.943585] 'ReglalistaAtacantesACK'	eth0		00:...	72....	17...	443	6032	TCP	64...
6	May 8 ...	[244842.137552] 'ReglalistaAtacantesACK'	eth0		00:...	72....	17...	443	6032	TCP	64...
7	May 8 ...	[244842.288944] 'ReglalistaAtacantesACK'	eth0		00:...	72....	17...	443	6032	TCP	64...
8	May 8 ...	[244842.289320] 'ReglalistaAtacantesACK'	eth0		00:...	72....	17...	443	6032	TCP	64...
9	May 8 ...	[244848.608550] 'ReglalistaAtacantesACK'	lo		00:...	12...	12...	1	5933	TCP	0
10	May 8 ...	[244848.608600] 'ReglalistaAtacantesACK'	lo		00:...	12...	12...	59...		TCP	4096
11	May 8 ...	[244848.608634] 'ReglalistaAtacantesACK'	lo		00:...	12...	12...	9	5933	TCP	0
12	May 8 ...	[244848.608657] 'ReglalistaAtacantesACK'	lo		00:...	12...	12...	59...		TCP	3072
13	May 8 ...	[244848.608693] 'ReglalistaAtacantesACK'	lo		00:...	12...	12...	3	5933	TCP	0
14	May 8 ...	[244848.608716] 'ReglalistaAtacantesACK'	lo		00:...	12...	12...	59...		TCP	1024
15	May 8 ...	[244848.608734] 'ReglalistaAtacantesACK'	lo		00:...	12...	12...	8	5933	TCP	0
16	May 8 ...	[244848.608770] 'ReglalistaAtacantesACK'	lo		00:...	12...	12...	59...		TCP	3072
17	May 8 ...	[244848.608803] 'ReglalistaAtacantesACK'	lo		00:...	12...	12...	5	5933	TCP	0
18	May 8 ...	[244848.608853] 'ReglalistaAtacantesACK'	lo		00:...	12...	12...	59...		TCP	3072
19	May 8 ...	[244848.608871] 'ReglalistaAtacantesACK'	lo		00:...	12...	12...	2	5933	TCP	0
20	May 8 ...	[244848.608908] 'ReglalistaAtacantesACK'	lo		00:...	12...	12...	59...		TCP	1024
21	May 8 ...	[244848.608926] 'ReglalistaAtacantesACK'	lo		00:...	12...	12...	6	5933	TCP	0
22	May 8 ...	[244848.608962] 'ReglalistaAtacantesACK'	lo		00:...	12...	12...	59...		TCP	2048
23	May 8 ...	[244848.609006] 'ReglalistaAtacantesACK'	lo		00:...	12...	12...	7	5933	TCP	0
24	May 8 ...	[244848.609043] 'ReglalistaAtacantesACK'	lo		00:...	12...	12...	59...	1	TCP	3072
25	May 8 ...	[244848.609061] 'ReglalistaAtacantesACK'	lo		00:...	12...	12...	10	5933	TCP	0
26	May 8 ...	[244848.609097] 'ReglalistaAtacantesACK'	lo		00:...	12...	12...	59...		TCP	1024
27	May 8 ...	[244848.609115] 'ReglalistaAtacantesACK'	lo		00:...	12...	12...	4	5933	TCP	0
28	May 8 ...	[244850.225474] 'ReglalistaAtacantesACK'	eth0		00:...	72....	17...	443	6032	TCP	64...
29	May 8 ...	[244850.225763] 'ReglalistaAtacantesACK'	eth0		00:...	72....	17...	443	6032	TCP	64...

Figura 57: Ventana con el contenido del Log

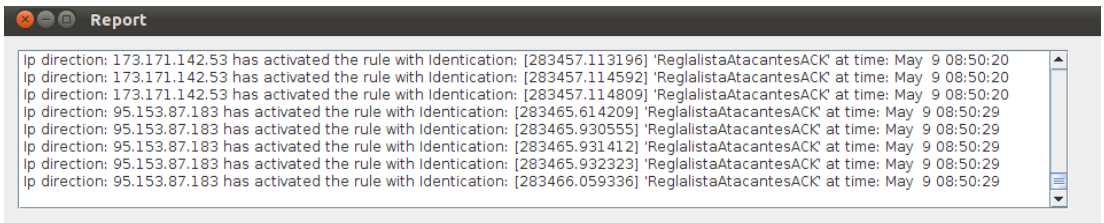


Figura 58: Ventana con el contenido del informe

10.7.13 CARGAR LA BASE DE DATOS

Esta opción permite al usuario cargar la base de datos en cualquier momento de la ejecución del programa. Para ello, basta con seleccionar la opción del menú "*Data Base"* y posteriormente seleccionar el submenú: "*Load DataBase"*.

En virtud de ello, la aplicación muestra en primer lugar un mensaje advirtiendo al usuario de que las reglas han sido cargadas con éxito.

Figura 59: Mensaje de base de datos cargada correctamente

A continuación, aparecerá una ventana, como la que se puede ver más abajo, listando todas las reglas cargadas de la base de datos.

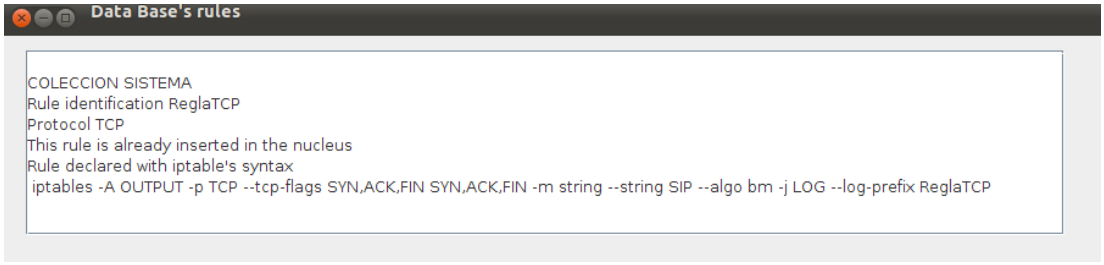


Figura 60: Ventana con el contenido de la base de datos actual

10.7.14 GUARDAR REGLAS EN LA BASE DE DATOS

Esta opción permite al guardar la configuración actual de las reglas en cualquier momento de la ejecución del programa. Para ello, basta con seleccionar la opción del menú "*Data Base"* y posteriormente seleccionar el sub menú: "*Save rules to Data Base"*.

En virtud de ello, la aplicación muestra en primer lugar, un mensaje de confirmación preguntando si está seguro de guardar las reglas en la base de datos. En caso afirmativo, la aplicación guardará las reglas en la base de datos de tal modo que si el usuario en otro momento quiere recuperarlas, sólo tendría que cargarlas de la base de datos.

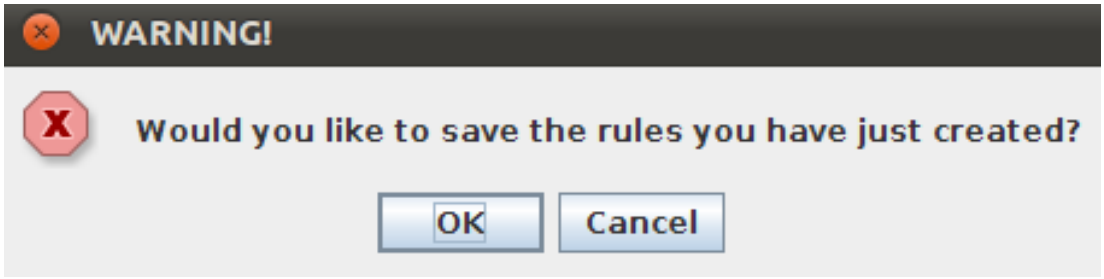


Figura 61: Mensaje de aviso para guardar las reglas

10.7.15 EXTRAER REGLA DEL FIREWALL POR ID

Esta opción permite al usuario extraer una regla cualquiera del *firewall*. Es necesario especificar el identificador con el que la regla se creó.

Para ello, el usuario deberá seleccionar la opción: *Iptables Management* y después el botón: "*Insert rule by rule ID"*.

A continuación la aplicación desplegará un campo de texto, como el que se puede apreciar en la figura inferior en el que el usuario debe introducir el identificador de la regla a extraer.

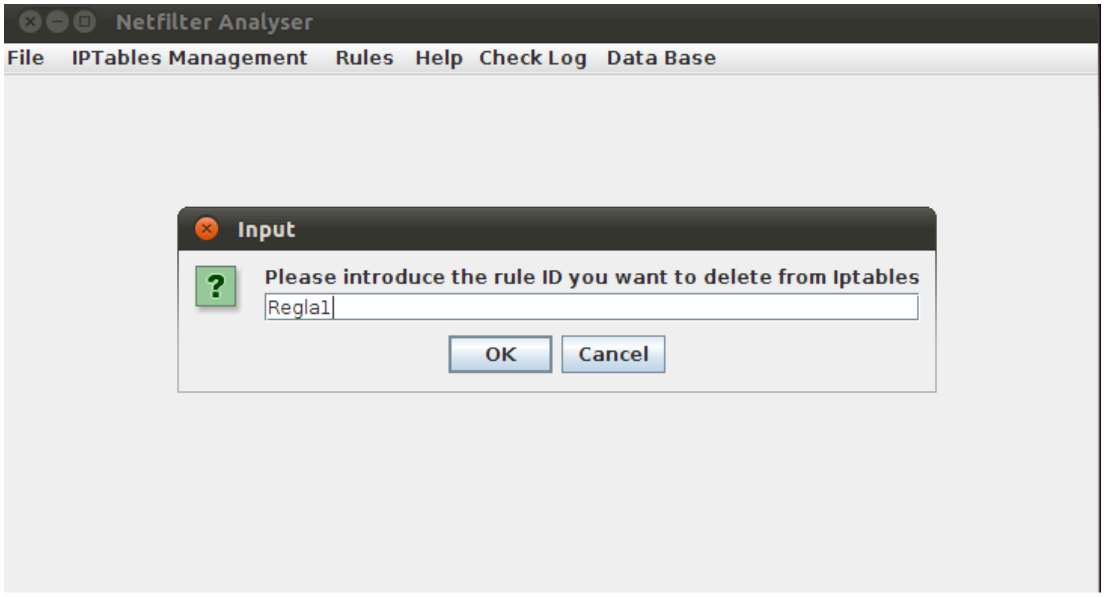


Figura 62: Ventana para introducir el nombre de la regla a extraer

En caso de que la operación haya sido efectuada con éxito, la aplicación informará de ello tal y como se puede apreciar en la figura inferior.

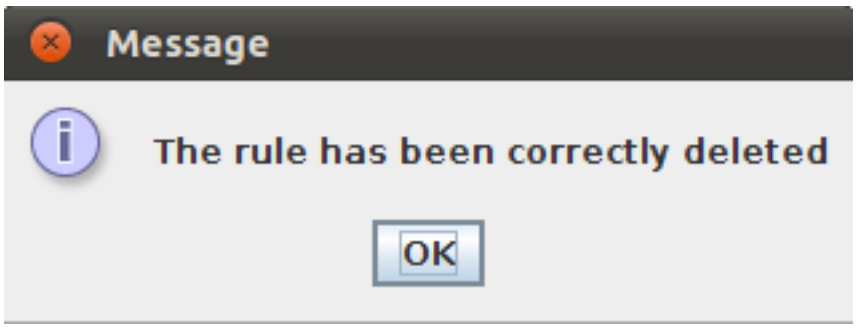


Figura 63: Mensaje de regla extraída correctamente

10.7.16 EXTRAER UNA COLECCIÓN DEL FIREWALL

La opción de extraer una colección del *firewall* es una opción muy útil para el usuario. Los pasos a seguir son similares a los de extraer una regla por ID.

En este caso, el usuario deberá introducir el identificador de la colección tal y como indica en la figura siguiente.

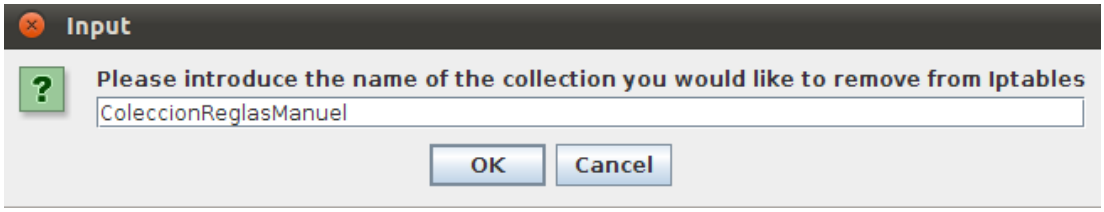


Figura 64: Ventana para introducir el nombre de la colección a extraer

En caso de que la operación haya sido efectuada con éxito la aplicación nos informará de ello.

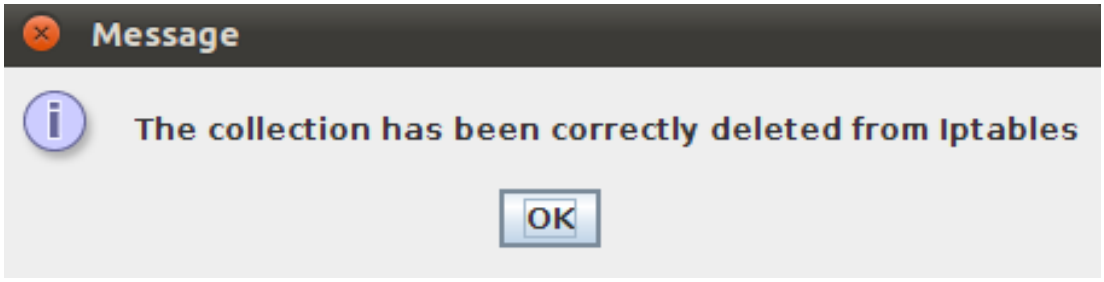


Figura 65: Mensaje de colección extraída correctamente

10.7.17 AYUDA

El botón *About us* nos informa de quién ha realizado la aplicación y en qué año se realizó. Para listar dicha información el usuario debe seleccionar la opción del menú *Help* y después el botón *About us.*

La información resultante al pulsar en dicho botón se aprecia a continuación:

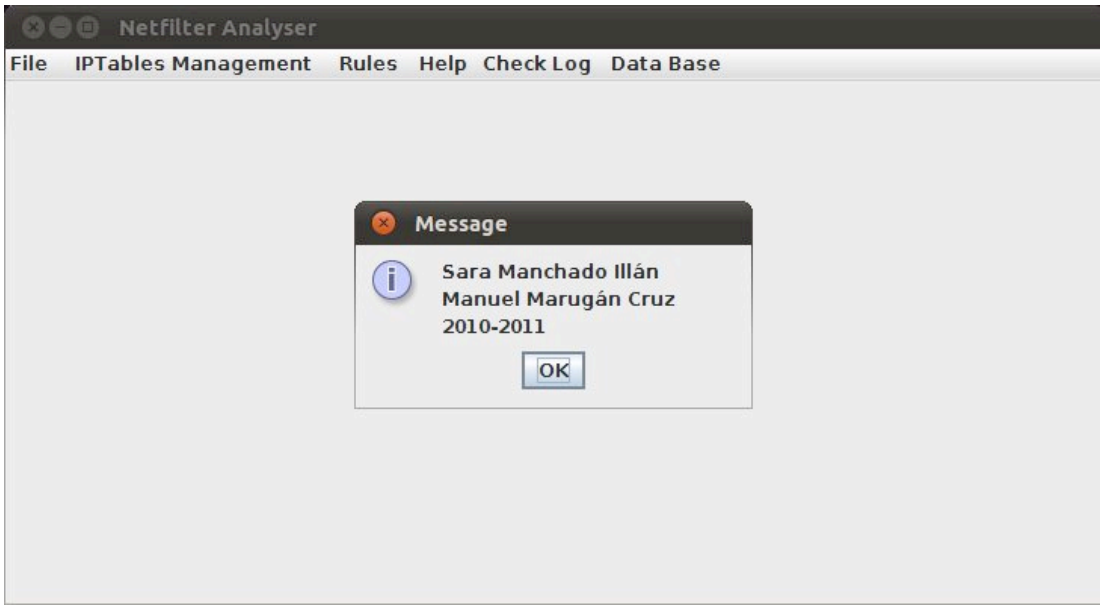


Figura 66: Mensaje sobre los desarrolladores de la aplicación

10.7.18 INICIO DE LA APLICACIÓN

Cuando el usuario ejecuta la aplicación, esta desplegará un mensaje preguntando al usuario si desea cargar la base de datos del sistema. Esta opción no es obligatoria ya que el usuario tiene la opción de cargar las reglas guardadas en la base de datos en cualquier momento de la ejecución de la aplicación.

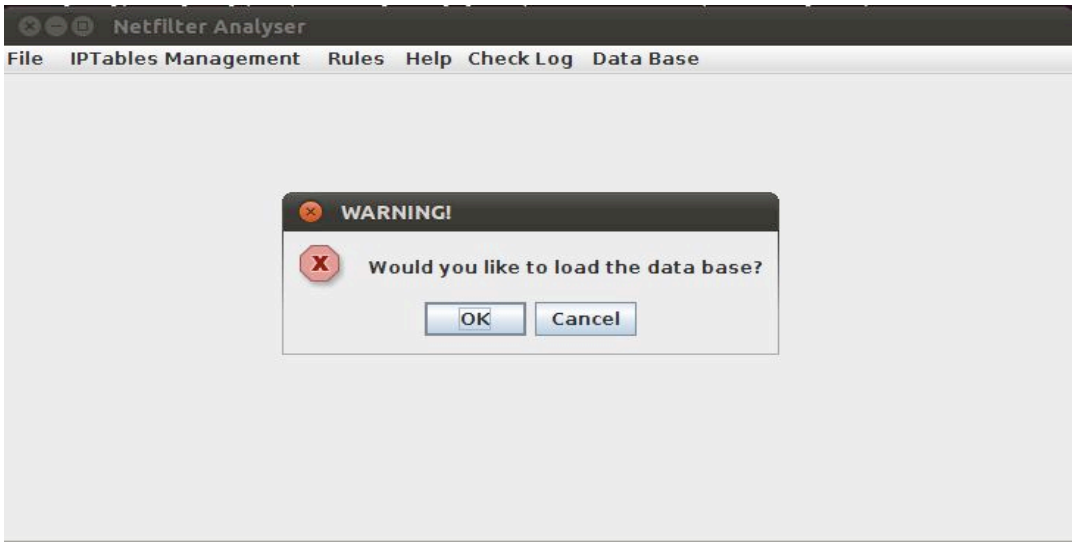


Figura 67: Mensaje de aviso al inicio de la aplicación

En caso de que el usuario haya optado por cargar la base de datos, la aplicación informará al usuario de las reglas que se van a insertar. Una ventana como la que se puede apreciar en la figura inferior muestra al usuario el estado de las reglas y su composición.

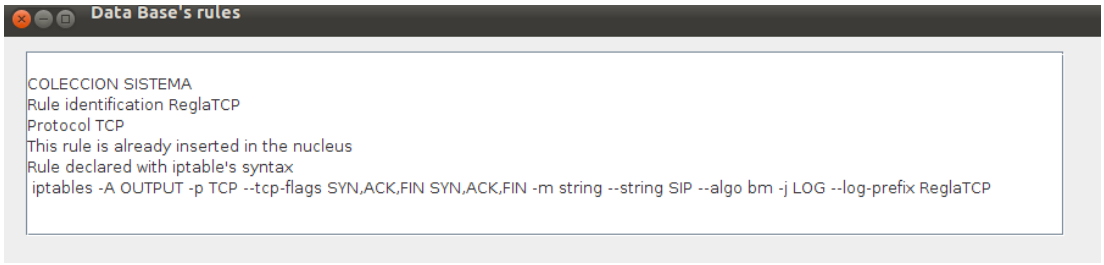


Figura 68: Ventana listando la base de datos cargada

10.7.19 CIERRE DE LA APLICACIÓN

Si el usuario desea abandonar la aplicación deberá seleccionar el botón del menú: *File* y después pulsar sobre la opción: "*Exit*".

La aplicación mostrará un mensaje de confirmación de salida como el siguiente:

Figura 69: Ventana de confirmación de cierre

La aplicación preguntará al usuario si está interesado en guardar los cambios acontecidos durante la ejecución del programa.

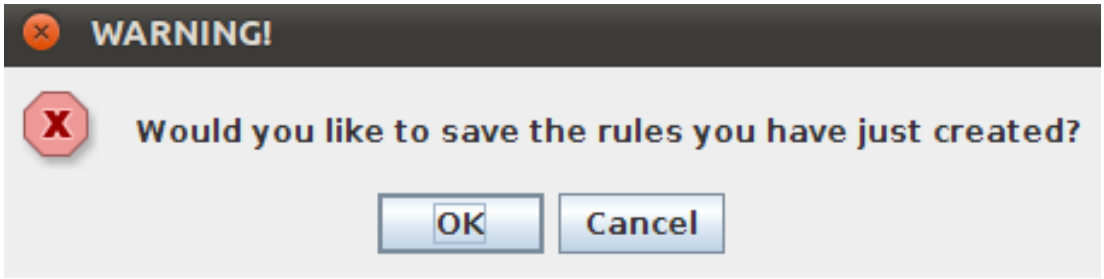


Figura 70: Ventana de confirmación de guardar cambios

Capítulo 10.

11. CONCLUSIONES

El principal objetivo del proyecto: disponer de una herramienta capaz de analizar el tráfico en la Red y generar alertas en base a comportamientos de tráfico reconocidos, ha sido en todo momento un objetivo ambicioso y motivador.

Cabe destacar, también, el hecho de que ser capaz de caracterizar de algún modo determinados comportamientos de tráfico supone un gran paso en el control del tráfico, uso de servicios y seguridad de las redes.

Respecto al desarrollo del proyecto, debemos mencionar que hemos tenido que afrontar distintos desafíos sin que éstos hayan afectado al alcance de los objetivos planteados inicialmente. El trabajar en máquinas virtuales sobre máquinas virtuales, hizo que el entorno de trabajo fuese un tanto inestable. La investigación de los patrones de tráfico, un campo en el que se requiere un alto grado de análisis y abstracción, ha constituido otro desafío para el desarrollo del proyecto.

A pesar de todo, se ha desarrollado una solución completa y funcional que cumple con los objetivos del proyecto.

Por otro lado, queremos destacar que el desarrollo de este proyecto ha supuesto una mayor y mejor comprensión de muchos conceptos, no sólo del ámbito de las redes de Internet sino también de las empresas que desarrollan sus negocios sirviéndose de ellas. Actualmente, las empresas son capaces de brindar muchos servicios haciendo uso de la infraestructura de las redes de Internet con lo que, tener un cierto control acerca del inmenso flujo de datos que circula a través de dichas redes se antoja fundamental. Dichas empresas están tremendamente interesadas en asegurar la fiabilidad de sus servicios y de tarificar según su uso. Cualquier agujero de seguridad en su infraestructura supone un agravio a su sistema de negocio provocando una gran pérdida de dinero y prestigio.

En base a estos supuestos, nuestro trabajo se basa en la búsqueda de un mayor conocimiento de los patrones de comportamiento de tráfico en la Red y podría resultar útil en el ámbito económico de ciertas empresas.

11.1 TRABAJO FUTURO

Durante el transcurso del proyecto han surgido nuevas líneas de interés para tener en cuenta en futuros trabajos. Nos gustaría destacar dos propuestas que consideramos interesantes.

En primer lugar, queremos referirnos a la conformación de tráfico de red. Dado que nuestro trabajo se basa en analizar y obtener los perfiles de un determinado comportamiento de red, podría resultar interesante usar dichos perfiles para facilitar la labor de la obtención de los recursos que proporciona la conformación de tráfico de red.

Por otro lado, la aplicación desarrollada deja algunos aspectos por mejorar. Dichos aspectos son la posibilidad de que la aplicación sea capaz de capturar nuevos comportamientos de tráfico tales como P2P u otro tipo de ataques. Otra mejora de la aplicación, que consideramos interesante, es la posibilidad de buscar una regla por su sintaxis.

11.2 REFLEXIÓN

Habiendo finalizado el proyecto nos sentimos satisfechos por el trabajo realizado y por alcanzar los objetivos planteados desde su comienzo.

Los retos que se plantearon inicialmente, tales como analizar el tráfico en la Red y detectar si se está produciendo alguno de los patrones de ataques indicados haciendo uso de la aplicación, han podido ser alcanzados con éxito.

Por otro lado, queremos indicar que, a pesar de dedicar tiempo y esfuerzo a estudiar herramientas como Snort, que finalmente no formaron parte del desarrollo del producto, dicho estudio nos ha enriquecido aportándonos conocimientos sobre *firewalls* e IDS pudiendo comprender sus diferencias y similitudes a un nivel muy detallado.

Además, ha sido una experiencia muy valiosa para el desarrollo profesional de los integrantes del proyecto ya que se han adquirido conocimientos técnicos pertenecientes a distintas ramas de la informática y se ha aprendido a planificar, gestionar y desarrollar un proyecto *software* por completo enmarcado en el ámbito de las redes.

Por último nos gustaría destacar el interés despertado en nosotros en el campo de las redes como consecuencia del trabajo e investigaciones realizadas.

APÉNDICE A. GLOSARIO DE TÉRMINOS Y ACRÓNIMOS

Sniffer: es un programa cuya función principal es capturar datos de la Red, sin interferir con la conexión a la que corresponden. Se utiliza principalmente para obtener contraseñas, información confidencial de la víctima o detectar comportamientos anómalos.

Firewall: también conocido como cortafuegos. Es un elemento *software* o *hardware* utilizados en las redes de ordenadores que bloquea accesos no deseados. La finalidad de un firewall es actuar como herramienta de seguridad al sistema.

Log: registro de paquetes de red durante un determinado periodo de tiempo. Un log es usado para registrar datos sobre paquetes que circulan por la red cumpliendo unas características determinadas.

Widget: es un tipo de aplicación que permite al usuario interactuar con la interfaz gráfica. Ejemplos de widgets son: ventanas, cajas de texto o *checkboxs*.

Interfaz gráfica de usuario: conocida también como GUI (*Graphical User Interface*) es un componente de una aplicación informática que utiliza un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz. Su propósito es hacer más fácil la interacción usuario-computador.

P2P (*Peer-to-Peer*)**:** es una red informática formada por un conjunto de nodos que se comportan al mismo tiempo como clientes y como servidores. Por ello, todos los nodos se comportan igual y pueden realizar las mismas operaciones. Algunos ejemplos de redes P2P son: *Napster*, *Kazaa* o *Emule*.

Router: es un dispositivo de interconexión de redes informáticas. Su función es realizar el encaminamiento de los paquetes o determinar la ruta que debe tomar el paquete de datos.

Códec: es una abreviatura de "comprensor/descompresor". Los códec son usados para la comprensión y descomprensión de los datos. Pueden ejecutarse en *software*, *hardware* o en ambos.

Hacker: persona con grandes conocimientos de informática que accede a sistemas informáticos de forma ilegal y no permitida para manipular cierta información.

DNAT (*Destination Network Address Translation*): se utiliza cuando la dirección IP es pública y se desea redirigir el acceso al firewall a otros computadores, es decir, cambia la dirección destino del paquete para poder redirigirlo a otra máquina.

SNAT (*Destination Network Address Translation*): se utiliza para cambiar la dirección IP origen del paquete.

Framework: es una arquitectura informática unificada que ofrece componentes como una librería, pero además provee de plantillas o esqueletos que definen el funcionamiento de las aplicaciones

Plugin: es un programa que incorpora una nueva funcionalidad o un servicio específico a un sistema más grande.

Modelo de referencia OSI: el modelo OSI está formado por siete capas:

- Capa Física: define las características físicas de la Red.
- Capa de Enlace de datos: direccionamiento físico.
- Capa de Red: determina la ruta e IP.(Direccionamiento lógico).
- Capa de transporte: conexión de extremo a extremo y detección de errores.
- Capa de sesión: provee mecanismos para iniciar, gestionar y finalizar una sesión entre procesos de aplicación.
- Capa de presentación: define el formato de los datos que maneja la capa de aplicación.
- Capa de aplicación: servicios de red a aplicaciones.

PSTN (*Public Switched Telephone Network*), en español RTB (Red Telefónica Básica), es una red que emplea la tecnología de conmutación de circuitos tradicional optimizada para realizar comunicaciones de voz en tiempo real. Esta red telefónica garantiza la calidad de servicio en las llamadas.

MTU (*Maximum Transfer Unit*): indica la cantidad máxima de datos que se pueden transmitir en una trama física.

Base de datos: conjunto de información relacionada que se encuentra almacenada en un sistema informático.

RFC (*Request For Comments*): conjunto de documentos que recogen propuestas de estándares en Internet.

BIBLIOGRAFÍA

- NORTHCUTT, S y NOVAK, J. *Detección de intrusos. Guía avanzada*. 2ª edición, Prentice Hall, [2001].
- CHAPPELL, L. *Advanced Network analysis techniques*. [2000]. Podbooks.com.
- FABERO JIMÉNEZ, Juan Carlos. Material de la asignatura: *Laboratorio de Redes*, capítulos 2-7, [2010].
- STALLING, Williams. *Comunicaciones y redes de computadores*, 7ª edición, [2004].
- GURLEY, R. *Intrusion Detection*, Macmillan Technical Publishing, [2000].
- CHESWICK, W. R.; BELLOVIN, S. M.; RUBIN, A. D. Firewall and Internet Security: Repelling the Wily Hacker, 2ª edición, Addison-Wesley Professional Computing, [2003].
- GOMEZ LOPEZ, Julio; GIL MONTOYA, Francisco. *VoIP y Asterik: redescubriendo la telefonía*, [2008].
- SCAMBRAY, J.; MCCLURE, S. Hacking Exposed: Network security secrets and solutions, 2ª edición, Osborne-McGraw Hill, [2001].
- BERSON, Tom. *Skype security evaluation*.
- User Mode Linux <http://user-mode-linux.sourceforge.net/>
- Snort<http://www.snort.org/>
  <http://www.adminso.es/images/d/d0/Pfc_Carlos_cap3.pdf>
- Iptables <http://www.netfilter.org/>
- Wireshark<http://www.wireshark.org/>
- Tcpdump<http://www.tcpdump.org/>
- Eclipse <http://www.eclipse.org/>
- Nmap<http://nmap.org/>