PROYECTO FIN DE CARRERA

Analizador de red (sniffer) en entorno GNU

Manuel Álvarez Crego_01/2005

TABLA DE CONTENIDOS

1. **Importancia del TFC dentro de la carrera**

El Trabajo fin de carrera (TFC) es una síntesis de los conocimientos adquiridos en la carrera. Es el primer trabajo profesional que realizamos y, por lo tanto, nos concede la oportunidad de desarrollar una actividad que de algún modo muestre nuestras actitudes.

Este proyecto trata de reflejar los conocimientos adquiridos en las asignaturas propias de la carrera, tal como *Fundamentos de programación, Sistemas operativos* y *Ampliación de sistemas operativos, Redes, Estructura de redes de computadores y Seguridad en redes de computadores.*

2. Plan de Trabajo

2.1. Tarea 1

- **Temporización:** 2 semanas (30 de septiembre 13 de octubre).
- **Descripción:** Recogida y clasificación de toda la información que pueda ser relevante para llevar a cabo el proyecto.
- **Objetivos:**
  - Tener una visión clara sobre la funcionalidad que tendrá la herramienta de análisis.
  - Tener una visión general sobre las herramientas más usadas en el momento para realizar escuchas en la red.

2.2. Tarea 2

- **Temporización:** 3 semanas (14 de octubre 2 de noviembre).
- **Descripción:** Análisis y diseño de la herramienta a implementar.
- **Objetivos:**
  - Establecer los requisitos que tendrá que tener la aplicación.
  - Obtener el análisis de la aplicación en función de los requisitos y de la funcionalidad que se le quiere dar.
  - Obtener el diseño de la aplicación a implementar, basándonos en el análisis del punto anterior.

2.3. Tarea 3

- **Temporización:** 6 semanas (3 de noviembre 15 de diciembre).
- **Descripción:** Implementación y juego de pruebas.
- **Objetivos:**
  - Implementar la aplicación en función del diseño obtenido. Generar los juegos de pruebas adecuados para garantizar el correcto funcionamiento de la aplicación, y para comprobar que se cubren los requisitos establecidos.

2.4. Tarea 4

- **Temporización:** 1 semana (16 al 24 de diciembre).
- **Descripción:** Documentación del producto.
- **Objetivos:**
  - Documentar el funcionamiento de la aplicación.

2.5. Tarea 5

- **Temporización:** 2 semanas (24 de diciembre 9 de enero).
- **Descripción:** Arreglos finales de la memoria y preparar la presentación de la herramienta que se ha creado.
- **Objetivos:**
  - Sintaxis de la memoria tomada durante el proyecto.
  - Crear una presentación en PowerPoint de la aplicación creada.

3. Introducción

El Departamento de Defensa de EE.UU. creó el modelo TCP/IP porque necesitaba una arquitectura que pudiera conectar múltiples redes y que tuviera la capacidad de mantener conexiones aun cuando una parte de la subred estuviese dañada por motivo de alguna guerra nuclear. Esto llevó a la creación del proyecto ARPANET (promovido y financiado por el DARPA, sección del Departamento de Defensa dedicada a la investigación). De este proyecto surgió el modelo de comunicación entre ordenadores de diferentes redes basado en el intercambio de paquetes.

Varias universidades americanas modificaron este modelo de comunicación, creando un sistema propio, llamado Internetting que, al irse ampliando a redes cada vez mayores, pasó a llamarse Internet. La base de esta Red de redes fue el modelo TCP/IP, el cual está basado en el tipo de red packet-switched (de conmutación de paquetes) y tiene cuatro capas: la capa de aplicación, la capa de transporte, la capa de Internet y la capa de red.

Como conclusión, con referencia a las redes, el TCP/IP tenía dos objetivos:

1) **construir** una interconexión de redes que proporcionase servicios de comunicación universales: una *red*, o *internet*.

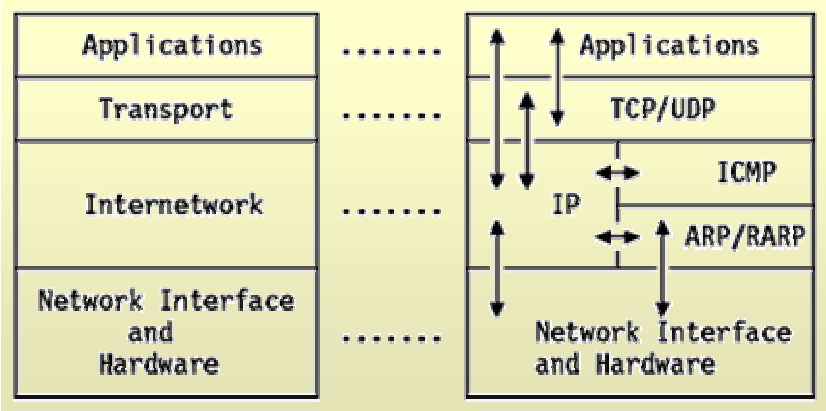
2) **interconectar** distintas redes físicas para formar lo que al usuario le parece una única y gran red. Tal conjunto de redes interconectadas se denomina *"internetwork"* o *internet*.

Ethernet es la tecnología más extendida en todo el mundo para la implementación de redes de área local. Gestiona el intercambio de datos entre ordenadores, pudiendo usar diferentes protocolos como TCP/IP, Netware, AppleTalk, VINES, etc. El más extendido es la pila de protocolos TCP/IP (Transport Control Protocol/Internet Protocol), que es un modelo práctico, a

nivel mundial, y un soporte para la intercomunicación de todo tipo de redes y sobre el que se ha desarrollado Internet.

El modelo de referencia TCP/IP y la pila de protocolo TCP/IP hacen que sea posible la comunicación entre dos ordenadores desde cualquier parte del mundo. La pila TCP/IP se llama así por dos de sus protocolos más importantes: TCP ("Transmisión Control Protocol") de IP ("Internet Protocol"). TCP/IP, como la mayoría del software de red, está modelado en capas. Esta representación conduce al término *pila de protocolos*.

Los protocolos de Internet se modelan en cuatro capas:



Modelo arquitectónico – Cada capa representa un ":q.package:eq." de funciones.

El protocolo de Ethernet trabaja enviando la información del paquete a todos los *hosts* en el mismo circuito. La cabecera del paquete contiene la dirección apropiada de la máquina destino. Solamente la máquina con la dirección que va en la cabecera se supone que acepta el paquete. Una máquina que está aceptando todos los paquetes, sin importar lo que ponga en la cabecera del paquete, se dice que está en modo promiscuo.

Un sniffer Ethernet es un software que trabaja en conjunto con la tarjeta de interfaz de la red (NIC, Network Interface Card) para absorber indiscriminadamente todo el tráfico que esté dentro del umbral de audición del sistema de escucha. Y no sólo el tráfico que vaya dirigido a una tarjeta de red,

sino a la dirección de difusión de la red 255.255.255.255 (es decir, a todas partes). Para ello el sniffer deberá hacer que la tarjeta de red entre en un estado especial denominado modo "promiscuo", en el que recibirá todos los paquetes que se desplazan por la red. La propia palabra promiscuo determina que escucha de todas partes.

Lo primero que hay que hacer es colocar la tarjeta de red afectada en modo promiscuo. Una vez que el hardware de la red se encuentre en modo promiscuo, el software del sniffer puede capturar y analizar cualquier tráfico que pase por el segmento local de Ethernet.

Sin embargo, las comunicaciones entre ordenadores consisten en datos binarios aparentemente al azar. Por lo tanto, los sniffers vienen con una característica conocida como el "análisis del protocolo", que les permite "descifrar" el tráfico del ordenador y hacer que tengan sentido todos esos datos binarios capturados de los paquetes.

3.1. Descripción de los campos de las cabeceras

3.1.1. Cabecera de una trama Ethernet

Preámbulo	Dir Destino	Dir Fuente	Tipo de Trama	Datos	CRC
64 bits	48 bits	48 bits	16 bits	368 - 12000 bits	32 bits

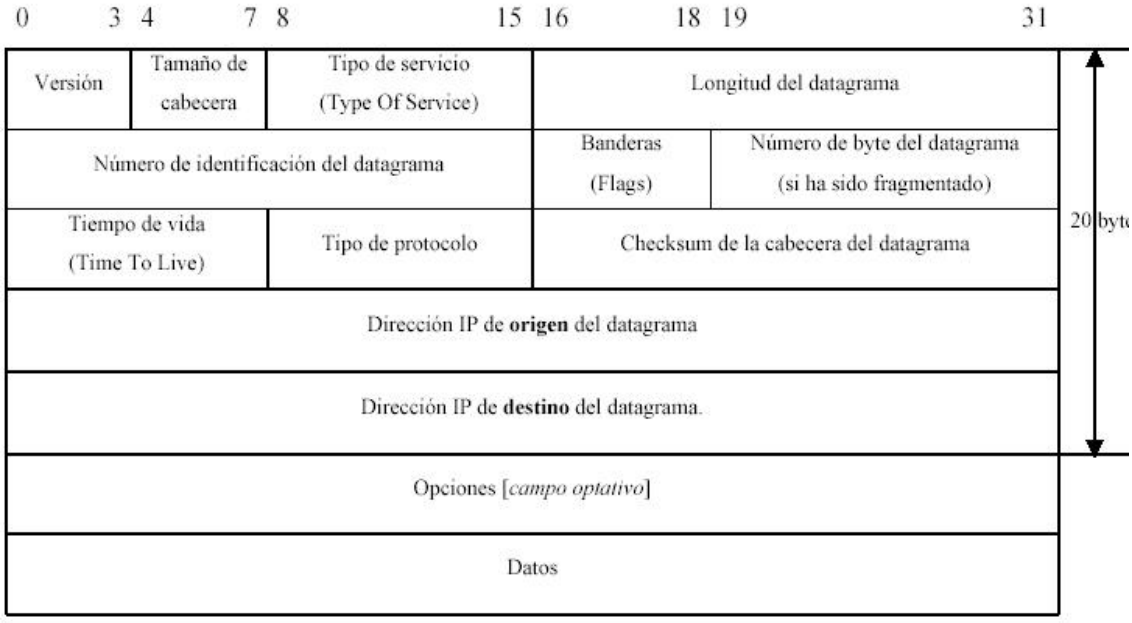
El **preámbulo** permite a nodos receptores sincronizarse (1's y 0's alternados).

El **CRC**(Cyclic Redundancy Check) sirve para detectar errores en la transmisión.

El **tipo de trama** se utiliza para saber el tipo de información que transporta la trama o el protocolo de nivel superior a utilizar, no necesariamente TCP/IP.

Tanto la **dirección de origen como la de destino** (MAC address), están formadas por dos números de 48 bits.

3.1.2. Cabecera de un datagrama IP



La **versión** (4 bits) sirve para identificar a qué versión específica (RFC) hace referencia el formato del datagrama. Esta información sólo es utilizada por los routers y capa IP de origen y final del datagrama. Esto permite la coexistencia de diferentes versiones del protocolo IP de una forma transparente al usuario. La versión actual es la 4 (conocida también cómo IPv4).

El **tamaño de la cabecera** (*Header Length*), son 4 bits (2^4 = 16 posiciones, 0...15) que indican el número de palabras de 32 bits que ocupa la cabecera. Estos 4 bits de tamaño máximo nos limitan a un tamaño de cabecera máximo de 60 bytes (15 * 32 bits)= 60 bytes). No obstante, el valor usual de este campo es 5 (5 * 32 bits = 20 bytes).

El **campo del tipo de servicio** (*Type Of Service*), se compone de 8 bits. Los primeros 3 bits tienen una función obsoleta y no se contemplan actualmente. Los 4 bits siguientes definen el tipo de servicio (ver figura 2-12) y el último bit no se utiliza actualmente y debe tener valor 0. Sólo 1 de los 4 bits del tipo de servicio puede estar activo a la vez. El tipo de servicio determina la política a

seguir en el envío del datagrama por Internet. Las opciones posibles son:

1. minimizar el retraso (*minimize delay*)
2. maximizar el rendimiento (*maximize throughput*)
3. maximizar la fiabilidad del transporte (*maximize reliability*)
4. minimizar el coste económico del transporte (*minimize monetary cost*).

La **longitud del datagrama** (*Total Length*), es un número de 16 bits (2^{16} = 65536, o...65535) que indica la longitud total del datagrama. Este valor es muy importante, ya que nos permite saber qué tamaño de memoria debemos reservar para la recepción del datagrama. Además, nos indica el número de bytes a leer, lo que nos permite un simple control de error. De esta forma, si el valor es incorrecto, el número de bytes leídos será como máximo de 65535, acotando el error. Además nos limita el número de bytes a enviar en un datagrama (*Maximum Transfer Unit, MTU*) a 65535 – 20 (tamaño típico de la cabecera) = 65515 bytes. Si el tamaño del datagrama, es mayor que el tamaño máximo del paquete de red (Ej. Datagrama de 32000 bytes enviado sobre una Ethernet, que tiene un tamaño máximo de paquete de 1500 bytes), éste se fragmenta en N trozos.

El **número de identificación del datagrama** (*Identificartion Field*), es un número de 16 bits que, en caso de fragmentación de un datagrama, nos indica su posición en el datagrama original. Esto nos permite recomponer el datagrama original en la máquina de destino. Este valor nos indica que un datagrama puede ser fragmentado en un máximo de 65535 fragmentos.

Las **banderas** (*Flags*) son 3 bits. El primero permite señalar si el datagrama recibido es un fragmento de un datagrama mayor, bit M (*More*) activado. El segundo especifica si el datagrama no debe fragmentarse, bit DF (*Don't fragment*) activado, y el tercero no se utiliza actualmente, asignándole el valor 0.

El **tiempo de vida** (*Time To Live*), es un campo de 8 bits que indica el tiempo

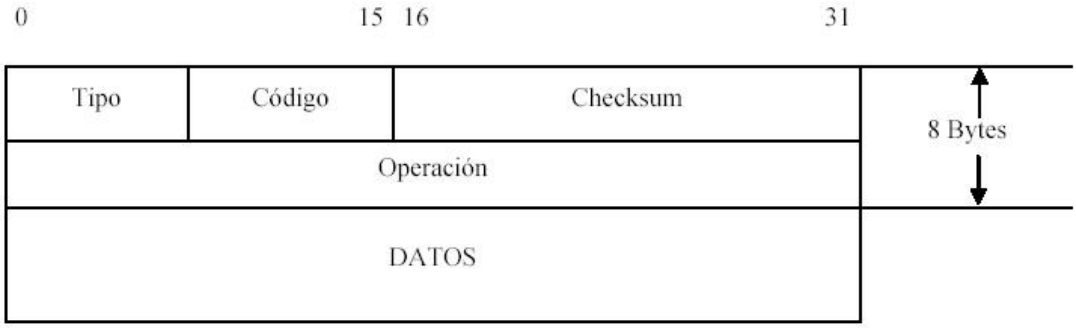
máximo que el datagrama será válido y podrá ser transmitido por la red. Esto permite un mecanismo de control para evitar datagramas que circulen eternamente por la red (por ejemplo, en el caso de bucles). Este campo se inicializa, en el ordenador de origen, a un valor (máximo 2^8 = 256) y se va decrementando en una unidad, cada vez que atraviesa un router. De esta forma, si se produce un bucle y/o no alcanza su destino en un máximo de 255 "saltos", es descartado. En este caso, se envía un datagrama ICMP de error al ordenador de origen para avisar de su pérdida.

El **tipo de protocolo** (*Protocol*) es un valor que indica a qué protocolo pertenece el datagrama (TCP, UDP, ICMP...). Es necesario, debido a que todos los servicios de Internet utilizan IP como transporte, lo cual hace necesario un mecanismo de discriminación entre los diferentes protocolos.

El **checksum de la cabecera del datagrama** (*Header Checksum*) es una suma de comprobación que afecta sólo a la cabecera del datagrama IP. El resto de protocolos TCP, UDP, IGMP... tienen su propia cabecera y *checksum*. Su función es simplemente la de un mecanismo de control de errores. De esta forma, si se encuentra un error en el *checksum* de un datagrama IP, éste es simplemente descartado y no se genera ningún mensaje de error. Esto implica que es deber de las capas superiores el control del flujo de los datagramas para asegurarse que éstos lleguen correctamente al destino, ya sea utilizando un protocolo fiable (TCP) o implementando internamente algún tipo de control.

Tanto la **dirección IP de origen como la de destino** (*IP address*) están formadas por dos números de 32 bits.

3.1.3. Cabecera de un mensaje ICMP

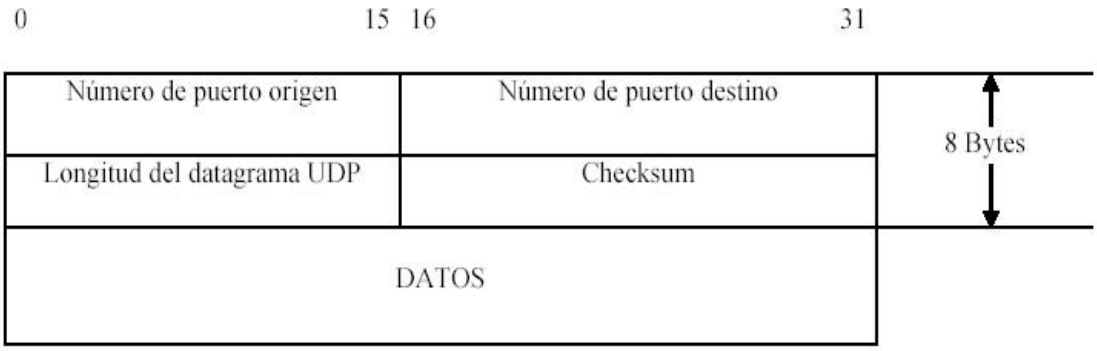


El **Tipo** (*Type*) indica el carácter del mensaje enviado, ya que el protocolo permite especificar una gran variedad de errores o mensajes de control de flujo de las comunicaciones.

El campo de **Código** (*Code*) indica el código de error dentro del tipo de error indicado en el campo "tipo". Es decir, agrupa los mensajes en tipos y dentro de cada tipo especifica el código concreto al que se refiere.

El **Checksum** simplemente permite verificar la integridad del mensaje enviado, lo que permite detectar posibles errores en el envío, transporte o recepción del mensaje de control ICMP.

3.1.4. Cabecera de un datagrama UDP

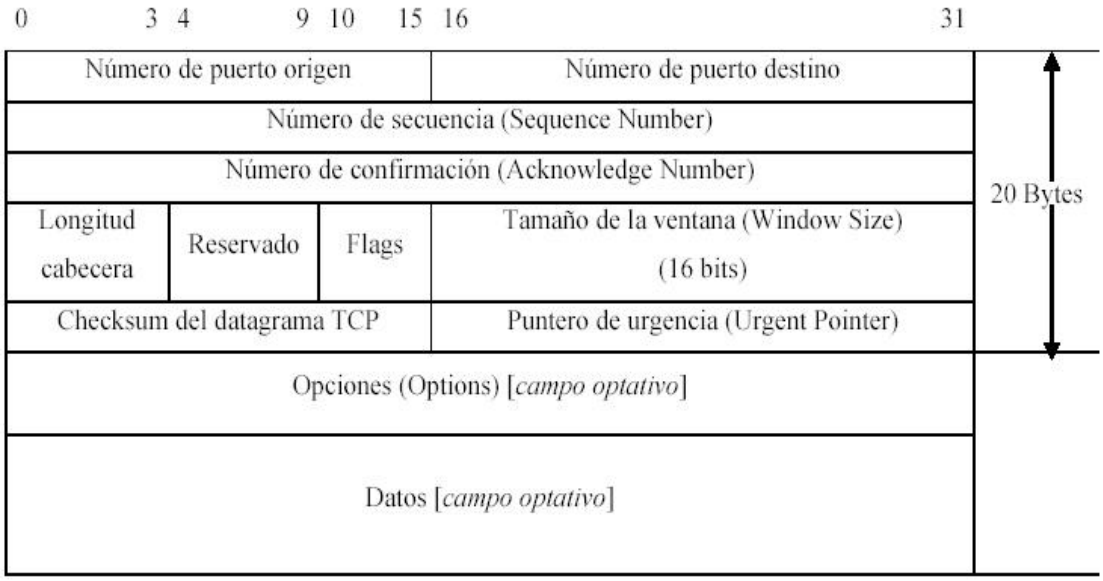


El **número de puerto** (*Port*) se utiliza en la comunicación entre dos ordenadores para diferenciar las diferentes conexiones existentes. Si tenemos varias comunicaciones desde nuestro ordenador (por ejemplo un TELNET al *cc.uab.es* mirando el correo y un FTP a *blues.uab.es* bajando un fichero), al recibir un datagrama IP debemos saber a cual de las conexiones pertenece. Asignando un número de puerto a la comunicación podemos saber a qué conexión pertenece. Al ser un número de 16 bits, podemos deducir que el número máximo de conexiones que un ordenador puede tener simultáneamente en uso es de 65535 (2^{16}).

La **longitud del datagrama** (*UDP Length*) hace referencia al tamaño del datagrama en bytes y engloba la cabecera (8 bytes) más los datos que transporta. El mínimo valor de longitud es 8 bytes (por lo tanto, el protocolo permite enviar un datagrama UDP con 0 bytes). Este campo es redundante, ya que utiliza IP para su transporte, y éste ya incorpora un campo para la longitud de los datos (ver figura 2-11), que sería la longitud del datagrama IP menos el tamaño de la cabecera.

El campo de **checksum,** al igual que en IP, sirve como método de control de los datos, verificando que no han sido alterados. Este *checksum* cubre tanto la cabecera UDP como los datos enviados. Es necesario, debido a que el *checksum* del protocolo IP tan sólo cubre la cabecera IP y no los datos que transporta. Si se detecta un error en el *checksum*, el datagrama es descartado sin ningún tipo de aviso.

3.1.5. Cabecera de un segmento TCP



El número de **puerto origen** y número de **puerto destino** sirven para diferenciar una comunicación en un ordenador de las demás. Cumple la misma función que en el datagrama UDP.

La tupla formada por la dirección IP y el número de puerto se denomina *socket*. Este término se utilizó luego en la especificación de la interface de programación de Berkeley (API). Análogamente, la agrupación de las dos tuplas que definen una conexión entre dos ordenadores se denomina *socket pair*.

El **número de secuencia** (*Sequence Number*) identifica el byte concreto del flujo de datos que actualmente se envía del emisor al receptor. De esta forma, TCP numera los bytes de la comunicación de una forma consecutiva, a partir del número de secuencia inicial. Cuando se establece una comunicación, emisor y receptor eligen un número de secuencia común, lo que nos permite implementar mecanismos de control como asegurar que los datos lleguen en el orden adecuado. Es un número de 32 bits, con lo que podemos enviar 2^{32}-1 bytes antes de que reinicie el ciclo.

El **número de confirmación** (*Acknowledge Number*) es el número de secuencia más uno. De este modo, se especifica al emisor que los datos enviados hasta este número de secuencia menos uno son correctos. De aquí la importancia de la elección al principio de la comunicación de un número de secuencia común.

La **longitud de la cabecera** (*header Length*) especifica en palabras de 32 bits (4 bytes) el tamaño de la cabecera del segmento TCP, incluyendo las posibles opciones. De esta forma, el tamaño máximo es 15 * 4 = 60 bytes. No obstante, lo usual es tener un tamaño de 20 bytes (cabecera donde no se incluyen opciones).

Las **banderas** (*Flags*) son las encargadas de especificar los diferentes estados de la comunicación. Asimismo, también validan los valores de los distintos campos de la cabecera de control. Puede haber simultáneamente varios *flags* activados. En la figura 1- 13 podemos ver los distintos *flags* existentes y su significado.

El **tamaño de la ventana** (*Window Size*) es el número de bytes, desde el número especificado en el campo de confirmación, que el receptor está dispuesto a aceptar. El tamaño máximo es de (2^{16}) 65535 bytes. De esta forma, el protocolo TCP permite la regulación del flujo de datos entre el emisor y el receptor.

El **checksum del segmento** TCP, al igual que el del UDP o IP, tiene la función de controlar los posibles errores que se produzcan en la transmisión. Este *checksum* engloba la cabecera TCP y los datos. En caso de error, el datagrama/segmento queda descartado y el propio protocolo es el encargado de asegurar la retransmisión de los segmentos erróneos y/o perdidos.

El **puntero de urgencia** (*Urgent Poiner*) es válido sólo si el *flag* de **URG** se encuentra activado. Consiste en un valor positivo que se debe sumar al número de secuencia, especificando una posición adelantada donde podemos enviar datos urgentes.

4. Analizadores de red con funcionalidades similares:

4.1. Linsniffer

Linsniffer es sniffer sencillo y directo. Su propósito principal es capturar nombres de usuarios y contraseñas, y ésta es una función en la que sobresale.

Aplicación creada por Mike Edulla

Necesita: Archivos de cabecera C e IP.

Ubicación: http://downloads.securityfocus.com/tools/linsniffer.c

4.2. Linux_sniffer

Linux_sniffer ofrece una vista algo más detallada.

Aplicación: linux_sniffer por loq.

Necesita: archivos de cabecera C e IP

Ubicación: http://www.the-blue-orb.com/link/security/linux_sniffer/linux_sniffer.c

4.3. Hunt

Es otra opción útil cuando se necesita una salida menos compleja y más fácil de leer, un seguimiento de comandos más sencillo y snooping de sesiones.

Aplicación: hunt de Pavel Krauz.

Necesita: cabeceras de C e IP y Linux 2.0.35+, Glibc 2.0.7 con linuxThreads (o no).

Ubicación: http://packetstorm.linuxsecurity.com/sniffers/hunt/

4.4. Sniffit

Es un sniffer realmente potente. Su capacidad de configuración es muy alta, aunque su proceso de aprendizaje es laborioso.

Aplicación: Sniffit de Brecht Claerhout.

Necesita: cabeceras de C y de IR

Ubicación: http://reptile.rug.ac.be/~coder/sniffit/sniffit.html

4.5. Ethereal

Es un sniffer para Linux (y UNIX en general) que utiliza GUI y que ofrece algunos servicios interesantes. Uno de ellos es que la GUI de Ethereal permite examinar fácilmente los datos del sniffer, bien desde una captura en tiempo real, bien desde archivos de capturas tcpdump previamente generados. Todo ello, unido al continuo filtro para obtener una mejor exploración, así como la compatibilidad con SNMP y la capacidad para realizar capturas sobre Ethernet, FDDI, PPP y Token Ring estándar, hace que Ethereal sea una buena opción.

Necesita: GTK y libpcap.

Ubicación: http://ethereal.zing.org/.

5. Especificaciones

5.1. Funcionamiento general

Se trata de una aplicación que permite obtener y procesar todo el tráfico que se obtiene a través de la tarjeta de red. Está dotada de opciones de filtrado, tratamiento estadístico de la información, generación de logs... En particular, a nivel de enlace, se tratarán paquetes ethernet; a nivel de red, paquetes ip, y a nivel de transporte, paquetes tcp, udp e icmp.

Se trabaja en un entorno donde se dispone de una red Ethernet y donde está implementada la pila de protocolos TCP/IP. Dado que se trabaja con protocolos TCP/IP, utilizaremos la interfaz Socket.

La aplicación permite diferentes funcionalidades configurables en todo momento por el usuario. A continuación se detallan estas opciones:

- Opción **visionar / almacenar**: con esta opción se pretende que los paquetes procesados sean visionados por pantalla, almacenados en fichero o ambas a la vez.

- Opción **campos activos**: con esta opción se pretende poder indicar qué campos del paquete deben ser visionados y/o almacenados. Los campos a tratar incluyen a todos los campos de las cabeceras ethernet, ip, tcp, udp e icmp, así como el cuerpo del paquete.

- Opción **filtrar paquetes**: con esta opción se pretende procesar paquetes que cumplan ciertos criterios. En el caso que ningún filtro esté activo, se procesarán todos los paquetes recibidos. Los filtros que deben ser implementados son los siguientes:

- **Filtro1**: se procesan los paquetes recibidos cuyas direcciones físicas (MAC address) se encuentren dentro de un listado.
- **Filtro2**: se procesan los paquetes recibidos cuyas direcciones lógicas (IP address) se encuentren dentro de un listado.
- **Filtro3**: se procesan los paquetes recibidos cuyos puertos se encuentren dentro de un listado.

Una vez almacenados los datos en el fichero correspondiente mediante la aplicación 'buscaClaves', se puede realizar un filtrado final que nos permita ver solamente las líneas con posibles claves.

6. Decisiones de diseño

La aplicación nos permitirá mostrar por pantalla los paquetes, o almacenarlos en el fichero especificado, o bien visualizarlos por pantalla y almacenarlos en fichero al mismo tiempo.

Referente a la opción de almacenado en fichero (en el archivo que el usuario desee), es importante destacar que, si el fichero en el que se van a guardar los paquetes no existe, éste será creado, pero si el fichero ya existía (es decir, que contenía información referente a otros paquetes filtrados en una anterior ejecución), en ese caso los nuevos datos de los paquetes filtrados en la actual ejecución se concatenan al final del fichero, pudiendo así contrastar los resultados de varias ejecuciones. Con el fin de diferenciar las ejecuciones concatenadas, al principio del fichero se escribirá una cabecera, incluyendo la fecha y hora de la actual ejecución del sniffer.

Dado que el programa se ejecuta en modo de bucle infinito, para así recibir paquetes de forma indeterminada hasta que el usuario no decida finalizar con ctrl+c, es necesario establecer una correcta finalización del programa, ya que éste se aborta repentinamente al pulsar ctrl+c. Para ello, se ha realizado un cambio del controlador de ctrl+c, de forma que, cuando se pulse dicha combinación para acabar el programa, se invocará una función “controlador” que finalizará correctamente la ejecución, cerrando el fichero de almacenamiento de paquetes (en caso de haberse abierto), cerrando el socket creado inicialmente, restableciendo el controlador original de ctrl+c y finalizando la ejecución en último término.

Con el fin de agilizar las operaciones de filtrado y adquisición de los campos deseados de cada protocolo, así como para dinamizar este proceso y hacerlo

lo más flexible posible al usuario, se ha decidido entrar todos los datos en respectivos ficheros, que serán accedidos por la aplicación, extrayendo sus datos y almacenando la información en una "Cola", una estructura de datos dinámica que permitirá almacenar en memoria toda la información, una vez extraída de los archivos, y así dotar de una mayor eficiencia al proceso. De este modo, se dispondrá de una cola para cada uno de los filtros MAC, IP y puertos. Para almacenar los datos de los diferentes campos a mostrar, especificados en el fichero de configuración inicial, se creará una estructura de datos "Datos_Sniffer", que dispondrá de un campo para cada uno de los protocolos (Ethernet, IP, TCP, UDP y ICMP), siendo cada uno de estos campos una cola que almacenará los campos deseados para cada protocolo, obtenidos del fichero de configuración inicial.

La sintaxis de los diferentes ficheros:

- **Fichero de filtro de direcciones MAC**: estableciendo una dirección MAC en cada línea, se introducirán siguiendo la estructura: v1:v2:v3:v4:v5:v6, donde v1..v6 son enteros entre 0 y 255. Es importante destacar que estos valores serán introducidos en decimal y no en hexadecimal.

- **Fichero de filtro de direcciones IP**: estableciendo una dirección IP en cada linea, se introducirán siguiendo la estructura: v1.v2.v3.v4, donde v1..v4 son valores enteros decimales entre 0 y 255.

- **Ficheros de filtro de puertos**: estableciendo un puerto en cada línea, se introducirán los valores de cada puerto (entre 0 y 65535).

  Al activar cada una de las opciones de filtrado, los valores de cada fichero serán almacenados en su respectiva cola, y sólo se procesarán los paquetes que superen el filtro de comparar los respectivos campos origen y destino y que, al menos uno de los dos, coincida con alguno de los valores respectivos del filtro que se ejecuta. Hay resaltar que los filtros pueden coexistir sin ningún

inconveniente, de forma que se puede activar uno de ellos, dos de ellos o los tres. Al activar más de un filtro, éstos actuarán en forma de AND lógico, de manera que los paquetes que se procesen deberán superar todos los filtros activos.

- **Fichero de configuración inicial**: establece qué campos de cada protocolo serán visualizados y/o almacenados en fichero por el sniffer (siempre que superen los filtros que haya activos). Si de algún protocolo no se especifica ningún campo, ese protocolo no estará referenciado al mostrar el paquete. En este fichero se especificarán los campos de los protocolos Ethernet, IP, TCP, UDP y ICMP. Los valores posibles para cada uno de estos protocolos son los siguientes (se muestra la estructura de cómo debe ser el fichero de configuración, con delimitadores para cada protocolo y seguidamente los campos a tener en cuenta de ese protocolo concreto, uno por línea):

  1- Campos de la trama de red --

  ---

  dest

  source

  protocol

  2- Campos del protocolo IP --

  ---

  version

  hl

  tos

length

id

offset

ttl

protocol

checksum

dest

source

3- Campos del protocolo TCP --

---

dest

source

sequence

ack

header

window

checksum

urgent

4- Campos del protocolo UDP --

---

dest

source

length

checksum

5- Campos del protocolo ICMP --

---

type

code

checksum

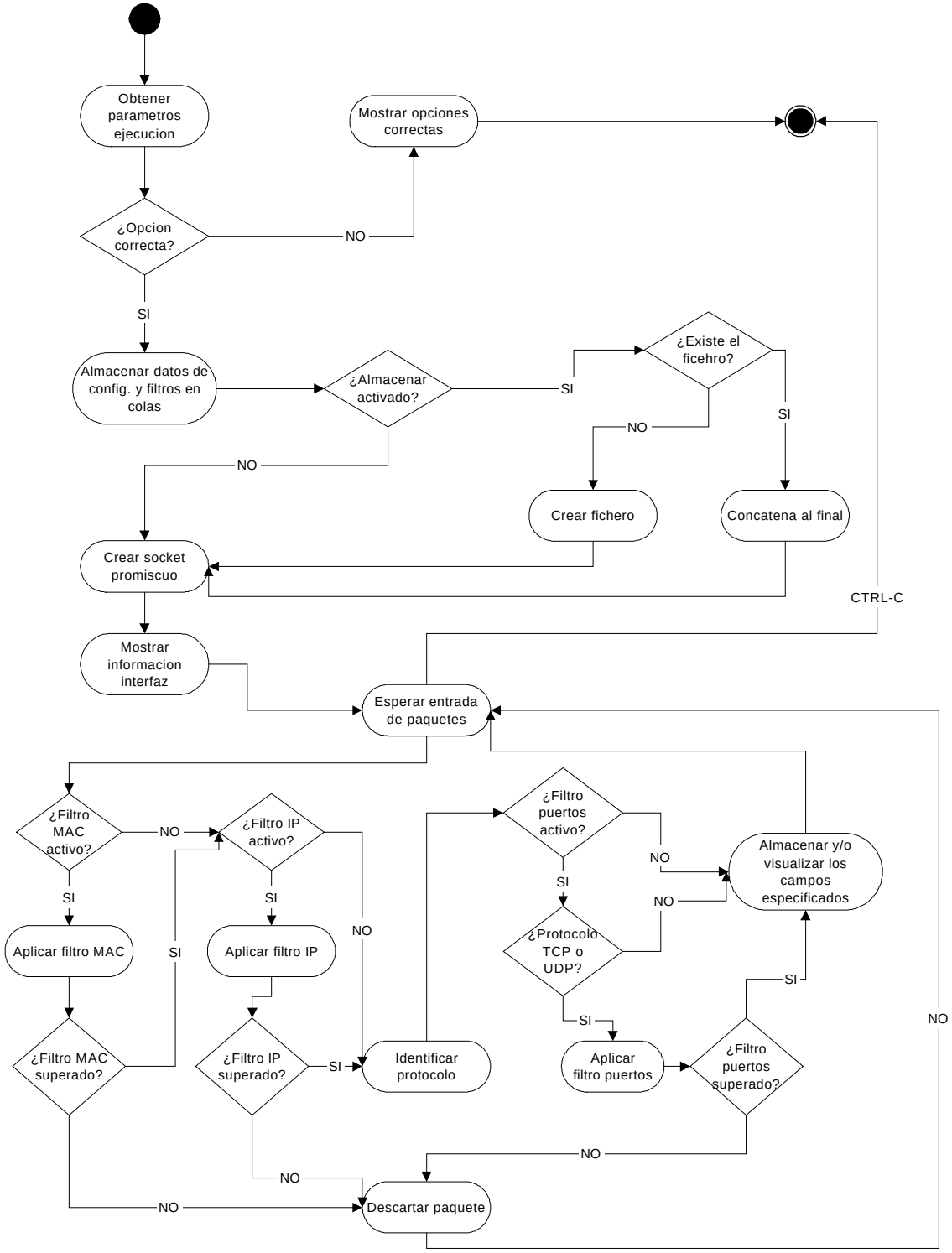
Para cada protocolo de este fichero, los campos introducidos (todos ellos son opcionales) se almacenarán en una cola diferente, englobadas en la estructura de datos “Datos_Sniffer”, que se utilizará en el momento de visualizar y/o almacenar, analizando cada campo del paquete procesado junto con los campos de Datos_Sniffer, decidiendo así si mostrar o no cada campo.

El proceso de visualización o almacenado se realiza una vez superados todos los filtros e identificados todos los protocolos del paquete recibido, y este proceso se hace de forma genérica, para optimizar al máximo el código. Es decir, no se replica el código para imprimir por pantalla y para almacenar por fichero sino que, utilizando “fprintf”, se almacena en “salida_datos”, pudiendo tomar el valor de “stdout” e imprimir por pantalla, o tomar el valor del puntero al fichero de almacenamiento, y así escribir en fichero. Si están activas a la vez las opciones de visualización y almacenado, se recorre dos veces dicho código, alternando el valor de “salida_datos”. Todos los campos de cada protocolo se mostrarán o no, en función de lo establecido en el fichero de configuración.

El usuario puede desear ver el contenido del cuerpo de los paquetes y, para ello, es necesario activar la opción '-d', mostrándose entonces los datos del paquete, tanto en pantalla como en fichero, aunque se ha decidido mostrar únicamente los caracteres imprimibles, a fin de no llenar con caracteres ilegibles la impresión al capturar paquetes con contenido binario.

La estructura del programa principal es la siguiente:

(se ha diseñado lo más robusta a fallos posible, de forma que cualquier situación anómala en los parámetros de ejecución o en el acceso incorrecto a los ficheros de filtrado o configuración es tratada e identificada)



- En primer término, se obtienen todos los parámetros de ejecución mediante 'getopt' y se aborta el programa si alguna opción es incorrecta.
- Se accede al fichero de configuración y se almacenan sus datos en las colas de 'Datos_Sniffer'.
- Se obtienen los datos de cada filtro si hay alguno activo.
- Se abre el fichero de almacenado en caso de activar esa opción, concatenando la información al final si ya existe el fichero o creándolo si no existía.
- Se crea el socket en modo promiscuo a partir de la interfaz de red proporcionada por parámetro.
- Se muestra toda la información de la interfaz actual, tal como: la dirección MAC de la interfaz, la dirección IP de la interfaz, la máscara de red de la interfaz y la dirección broadcast de la interfaz.
- Se ejecuta un bucle infinito, donde a cada iteración se realiza todo el procesado de un paquete, abortando mediante ctrl+c cuando se desee finalizar. Al iniciar la iteración, la ejecución se bloquea hasta que se recibe un paquete, entonces se obtiene el paquete como un array de caracteres.
- Se mapea el conjunto de bytes del paquete a una estructura de trama de red.
- Si el filtro de direcciones MAC está activo, se aplica: si se supera, se continúa el procesado del paquete actual y si no, se aborta la actual iteración y se accede a la lectura del siguiente paquete.
- Se mapea el conjunto de bytes del paquete a una estructura de paquete IP, saltando el tamaño ocupado por la estructura Ethernet anterior.

- Si el filtro de direcciones IP está activo, se aplica: si se supera, se continúa el procesado del paquete actual y si no, se aborta la actual iteración y se accede a la lectura del siguiente paquete.

- Se identifica el protocolo TCP, UDP o ICMP del paquete, a partir del campo 'protocol' de la cabecera IP, siendo valor 1 para ICMP, 6 para TCP y 17 para UDP.

- Se mapea el conjunto de bytes del paquete a la estructura TCP, UDP o ICMP correspondiente al protocolo identificado, sumando los tamaños de las cabeceras Ethernet e IP para ello.

- Si el filtro de puertos está activo, se aplica si el paquete tiene protocolo TCP o bien protocolo UDP: si se supera, se continúa el procesado del paquete actual y si no, se aborta la actual iteración y se accede a la lectura del siguiente paquete.

- Se visualizan y/o almacenan en ficheros los campos de cada protocolo especificados en el fichero de configuración, tal como se ha descrito anteriormente.

- Se finaliza la ejecución pulsando ctrl+c, invocando al controlador que permite acabar la ejecución de forma correcta.

Se han utilizado las estructuras de datos definidas en los archivos de cabecera de linux para las estructuras en las que se mapea la información del paquete a las diferentes cabeceras.

En cuanto a la estructuración de las diferentes partes del programa, el archivo 'sniffer.c' incluye el programa principal, la función de creación de socket promiscuo, dos funciones de conversión de direcciones y el controlador para ctrl+c.

El archivo 'logica.c' agrupa diversas funciones, como por ejemplo funciones de lectura de ficheros de filtro y configuración que extraen los datos y los

almacenan en colas, funciones de análisis y recorrido de las colas, funciones para aplicar los diferentes filtros, etc.

El archivo 'buscaClaves.c' se encarga de leer los datos guardados en el fichero de almacén y muestra por pantalla, y guarda en el fichero 'claves' sólo las líneas con posibles claves.

Finalmente, se dispone de los ficheros 'cola.c' y 'cola.h', donde 'cola.h' incluye la definición de las estructuras de datos 'Cola' y 'Datos_Sniffer', y 'cola.c' agrupa todas las funciones de la estructura de datos Cola, tales como creación de cola, adición de un elemento a la cola, supresión de un elemento de la cola, obtención del primer elemento, funciones que permiten recorrer la cola mediante punto de interés, etc.

7. Juego de Pruebas

Pruebas realizadas en una red local, donde la máquina (la que accede a la información, no la que actúa de sniffer) tiene la interfaz:

Interfaz: eth0

MAC: 00:90:27:5D:EF:D7

IP: 192.168.0.4

Netmask: 255.255.255.0

Broadcast: 192.168.0.255

1) Visualizar todos los campos de todos los paquetes sin filtros

./sniffer -r eth0 -c config.txt -v -a salida

Resultado relacionado con el acceso a la web "www.google.es" (tan sólo los 2 primeros paquetes):

| >> Tamaño total del paquete 62 Bytes

| :: Ethernet ::

| Dest 00:04:76:99:6A:92 | Src 00:90:27:5D:EF:D7 | Proto 8 |

| :: IP ::

| Version 4 | HL 5 | Tos 0 | Length 12288 |

| ID 508 | Fragment Offset 64 |

| TTL 128 | Proto 6 | Checksum 19440 |

| Src 192.168.0.4 |

| Dest 66.102.11.104 |

| :: TCP ::

| Src 1061 | Dest 80 |

| Sequence 35 |

| Ack Sequence 0 |

| Header 0 7 0 1 0 0 0 0 0 0 | Window 8192 |

| Checksum 34913 | Urgent Pointer 0 |

| >> Tamaño total del paquete 62 Bytes

| :: Ethernet ::

| Dest 00:90:27:5D:EF:D7 | Src 00:04:76:99:6A:92 | Proto 8 |

| :: IP ::

| Version 4 | HL 5 | Tos 0 | Length 12288 |

| ID 0 | Fragment Offset 64 |

| TTL 62 | Proto 6 | Checksum 20014 |

| Src 66.102.11.104 |

| Dest 192.168.0.4 |

| :: TCP ::

| Src 80 | Dest 1061 |

| Sequence 29101 |

| Ack Sequence 35 |

| Header 0 7 0 1 0 0 1 0 0 0 | Window 5840 |

| Checksum 28764 | Urgent Pointer 0 |

2) Resultado después de realizar un traceroute para comprobar la captura de paquetes UDP (en windows tracert):

./sniffer -r eth0 -c config.txt -v -a salida

C:\WINDOWS>tracert 192.168.0.3

Resultado (uno de los paquetes de interés):

| >> Tamaño total del paquete 92 Bytes

| :: Ethernet ::

| Dest 00:90:27:A2:16:15 | Src 00:90:27:5D:EF:D7 | Proto 8 |

| :: IP ::

| Version 4 | HL 5 | Tos 0 | Length 19968 |

| ID 542 | Fragment Offset 0 |

| TTL 128 | Proto 17 | Checksum 17819 |

| Src 192.168.0.4 |

| Dest 192.168.0.3 |

| :: UDP ::

| Src 137 | Dest 137 |

| Length 58 | Checksum 15925 |

3) Ping a www.google.es para ver paquetes ICMP.

./sniffer -r eth0 -c config.txt -v -a salida

C:\WINDOWS>ping www.google.es

Resultado:

| >> Tamaño total del paquete 74 Bytes

| :: Ethernet ::

| Dest 00:04:76:99:6A:92 | Src 00:90:27:5D:EF:D7 | Proto 8 |

| :: IP ::

| Version 4 | HL 5 | Tos 0 | Length 15360 |

| ID 578 | Fragment Offset 0 |

| TTL 32 | Proto 1 | Checksum 17738 |

| Src 192.168.0.4 |

| Dest 66.102.11.104 |

| :: ICMP ::

| Type 2048 | Code 0 | Checksum 16988 |

| >> Tamaño total del paquete 74 Bytes

| :: Ethernet ::

| Dest 00:90:27:5D:EF:D7 | Src 00:04:76:99:6A:92 | Proto 8 |

| :: IP ::

| Version 4 | HL 5 | Tos 0 | Length 15360 |

| ID 578 | Fragment Offset 0 |

| TTL 242 | Proto 1 | Checksum 17528 |

| Src 66.102.11.104 |

| Dest 192.168.0.4 |

| :: ICMP ::

| Type 0 | Code 0 | Checksum 19036 |

*) A partir de ahora utilizamos filtrado de campos, especificamos en el archivo config.txt los campos siguientes:

1- Campos de la trama de red --

dest

source

2- Campos del protocolo IP --

dest

source

3- Campos del protocolo TCP --

dest

source

4- Campos del protocolo UDP --

dest

source

5- Campos del protocolo ICMP --

type

code

4) Filtro de direcciones MAC:

00:90:27:5D:EF:D7 -> 00:144:39:93:239:215 (en el filtro están en decimal)

./sniffer -r eth0 -c config.txt -m filtro1.txt -v -a salida

En el resultado se observa que sólo se muestran los paquetes que tienen como origen o destino la dirección MAC especificada en el filtro1.

Resultado:

| >> Tamaño total del paquete 72 Bytes

| :: Ethernet ::

| Dest 00:04:76:99:6A:92 | Src 00:90:27:5D:EF:D7 |

| :: IP ::

| Src 192.168.0.4 |

| Dest 195.235.113.3 |

| :: UDP ::

| Src 1035 | Dest 53 |

| >> Tamaño total del paquete 88 Bytes

| :: Ethernet ::

| Dest 00:90:27:5D:EF:D7 | Src 00:04:76:99:6A:92 |

| :: IP ::

| Src 195.235.113.3 |

| Dest 192.168.0.4 |

| :: UDP ::

| Src 53 | Dest 1035 |

5) Filtro de direcciones IP:

213.4.130.210 -> www.terra.es

213.73.40.217 -> www.uoc.edu

./sniffer -r eth0 -c config.txt -i filtro2.txt -v -a salida

Después de haber navegado por diversas webs, se puede comprobar que sólo se capturan los paquetes que tienen como origen o destino las ips especificadas en el filtro2.

Resultado:

| >> Tamaño total del paquete 62 Bytes

| :: Ethernet ::

| Dest 00:04:76:99:6A:92 | Src 00:90:27:5D:EF:D7 |

| :: IP ::

| Src 192.168.0.4 |

| Dest 213.4.130.210 |

| :: TCP ::

| Src 1075 | Dest 80 |

| >> Tamaño total del paquete 62 Bytes

| :: Ethernet ::

| Dest 00:90:27:5D:EF:D7 | Src 00:04:76:99:6A:92 |

| :: IP ::

| Src 213.4.130.210 |

| Dest 192.168.0.4 |

| :: TCP ::

| Src 80 | Dest 1075 |

| >> Tamaño total del paquete 1514 Bytes

| :: Ethernet ::

| Dest 00:90:27:5D:EF:D7 | Src 00:04:76:99:6A:92 |

| :: IP ::

| Src 213.73.40.217 |

| Dest 192.168.0.4 |

| :: TCP ::

| Src 80 | Dest 1073 |

| >> Tamaño total del paquete 458 Bytes

| :: Ethernet ::

| Dest 00:04:76:99:6A:92 | Src 00:90:27:5D:EF:D7 |

| :: IP ::

| Src 192.168.0.4 |

| Dest 213.73.40.217 |

| :: TCP ::

| Src 1074 | Dest 80 |

6) Filtro de puertos

110 -> POP3

23 -> TELNET

80 -> HTTP

./sniffer -r eth0 -c config.txt -p filtro3.txt -v -a salida

Resultado:

| >> Tamaño total del paquete 62 Bytes

| :: Ethernet ::

| Dest 00:04:76:99:6A:92 | Src 00:90:27:5D:EF:D7 |

| :: IP ::

| Src 192.168.0.4 |

| Dest 213.73.40.217 |

| :: TCP ::

| Src 1110 | Dest 23 |

| >> Tamaño total del paquete 62 Bytes

| :: Ethernet ::

| Dest 00:04:76:99:6A:92 | Src 00:90:27:5D:EF:D7 |

| :: IP ::

| Src 192.168.0.4 |

| Dest 213.4.130.210 |

| :: TCP ::

| Src 1115 | Dest 110 |

7) Múltiples filtros: MAC, IP y Puertos

00:90:27:5D:EF:D7 -> 00:144:39:93:239:215

213.4.130.210 -> www.terra.es

213.73.40.217 -> www.uoc.ed

110 -> POP3

23 -> TELNET

80 -> HTTP

./sniffer -r eth0 -c config.txt -m filtro1.txt -i filtro2.txt -p filtro3.txt -v -a salida

Como resultado, sólo se mostrarán los paquetes que superen los tres filtros.

Resultado:

| >> Tamaño total del paquete 62 Bytes

| :: Ethernet ::

| Dest 00:04:76:99:6A:92 | Src 00:90:27:5D:EF:D7 |

| :: IP ::

| Src 192.168.0.4 |

| Dest 213.4.130.210 |

| :: TCP ::

| Src 1118 | Dest 110 |

8) Captura de datos utilizando el filtro3 del punto anterior.

./sniffer -r eth0 -c config.txt -p filtro3.txt -v -a salida –d

8.1) Ejecutado en 192.168.0.4:

C:\WINDOWS>telnet pop.terra.es 110

Trying 213.4.129.129...

Connected to smtp.terra.es.

Escape character is '^]'.

+OK Bienvenido al servidor POP3 de TERRA

user eos123.terra.es

+OK password required for user eos123.terra.es

pass mac2005

+OK Maildrop ready

Resultado:

| >> Tamaño total del paquete 96 Bytes

| :: Ethernet ::

| Dest 00:90:27:5D:EF:D7 | Src 00:04:76:99:6A:92 |

| :: IP ::

| Src 213.4.129.129 |

| Dest 192.168.0.4 |

| :: TCP ::

| Src 110 | Dest 1124 |

CUERPO DEL PAQUETE (54 bytes cabeceras,96 bytes totales):

+OK Bienvenido al servidor POP3 de TERRA

| >> Tamaño total del paquete 102 Bytes

| :: Ethernet ::

| Dest 00:90:27:5D:EF:D7 | Src 00:04:76:99:6A:92 |

| :: IP ::

| Src 213.4.129.129 |

| Dest 192.168.0.4 |

| :: TCP ::

| Src 110 | Dest 1124 |

CUERPO DEL PAQUETE (54 bytes cabeceras,102 bytes totales):

+OK password required for user eos123.terra.es

| >> Tamaño total del paquete 68 Bytes

| :: Ethernet ::

| Dest 00:04:76:99:6A:92 | Src 00:90:27:5D:EF:D7 |

| :: IP ::

| Src 192.168.0.4 |

| Dest 213.4.129.129 |

| :: TCP ::

| Src 1127 | Dest 110 |

CUERPO DEL PAQUETE (54 bytes cabeceras,68 bytes totales):

---

pass mac2004

8.2)Entramos en la página web "http://www.telefonica.net/correo" para acceder a nuestro correo:

ID eos123@telefonica.net

Contraseña mac2005

Resultado (entre otros muchos paquetes):

| >> Tamaño total del paquete 547 Bytes

| :: Ethernet ::

| Dest 00:04:76:99:6A:92 | Src 00:90:27:5D:EF:D7 |

| :: IP ::

| Src 192.168.0.4 |

| Dest 213.4.129.131 |

| :: TCP ::

| Src 1171 | Dest 80 |

CUERPO DEL PAQUETE (54 bytes cabeceras,547 bytes totales):

POST /tenet/login_telefonica.cgi HTTP/1.1

Accept: */*

Referer: http://telefonica.terra.es/correo/

Accept-Language: es

Content-Type: application/x-www-form-urlencoded

Accept-Encoding: gzip, deflate

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows 98)

Host: correo.terra.es

Content-Length: 76

Connection: Keep-Alive

Cache-Control: no-cache

Cookie: lubid=0101FC27B98B500016B141652ADD0000010A00000000

user=eos123&password=mac2005&dominio=telefonica.net&image22.x=31&image22.y=7

9)Localizar líneas con altas posibilidades de contener claves. Una vez guardados todos los datos anteriores en el fichero "salida", ejecutamos:

./buscaClaves salida

Resultado:

Líneas con posibles claves:

+OK password required for user eos123.terra.es

pass mac2005

user=eos123&password=mac2005&dominio=telefonica.net&image22.x=31&image22.y=1

me=form>cmd<input name=cmd>old<input name=oldPassword>new<input name=newPass

De todo el código capturado, nos hemos quedado tan sólo con cuatro líneas, de las cuales 3 muestran información de interés. Hemos conseguido capturar una clave de correo vía telnet (al puerto 110) y otra vía http.

8. Bibliografía

- “Unix, Programación avanzada” Fco. Manuel Márquez García. Editorial ra-ma.

- Utilización de los sockets en Linux:

http://www.elrincondelprogramador.com/default.asp?pag=articulos%2Fleer.asp&id=2 (Noviembre 2004)

http://www.arrakis.es/~dmrq/beej/index.html (Noviembre 2004)

- Descripción del funcionamiento de los conectores de paquetes (paquet socket):

http://www.linuxinfor.com/spanish/man7/packet.html (Diciembre 2004)

- Código fuente de varios sniffers con explicaciones de su funcionamiento:

http://packetstormsecurity.nl/sniffers/index6.html (Octubre 2004)

http://www.vilecha.com/Autodidactas/sniffers.html (Octubre 2004)

9. Anexo (Manual del programa)

Para la compilación del programa, se ha creado un archivo “Crear_Ejecutables” que realiza el proceso de compilación y lincado de los diferentes archivos de la aplicación. En el apartado de implementación se puede observar su contenido. Para compilar el programa basta pues ejecutar./Crear_Ejecutables, obteniendo como resultado los archivos ejecutables 'sniffer' y “buscaClaves”.

Los diferentes modos de ejecución del programa se especifican al ejecutar la aplicación, por medio de parámetros de ejecución. A continuación se detallan estos parámetros:

- **-r** <interfaz de red>: Se establece cuál va ser la interfaz de red (eth0, eth1, lo) sobre la que se conectará un socket en modo promiscuo para captar todos los paquetes.
- **-c** <fichero>: Aporta el fichero de configuración inicial en el que se explicitan cuáles son los campos de cada protocolo que se desean visualizar y/o almacenar.
- **-v** : Al activarla, se visualizarán por la salida estándar todos los paquetes recibidos con su respectiva información, en función del fichero de configuración.
- **-a** <fichero>: Permite almacenar en un fichero todos los paquetes procesados por el sniffer, del mismo modo que se visualizan por pantalla.
- **-m** <fichero>: filtro de direcciones MAC. Sólo se procesarán los paquetes cuya dirección MAC origen o destino coincida con alguna de las direcciones existentes en el archivo proporcionado.

- **-i** <fichero>: filtro de direcciones IP. Sólo se procesarán los paquetes cuya dirección IP origen o destino coincida con alguna de las direcciones del archivo.

- **-p** <fichero>: filtro de puertos. Sólo se procesarán los paquetes cuyos puertos origen o destino coincidan con alguno de los puertos incluidos en el archivo.

- **-d** : Esta opción permite mostrar los datos del cuerpo del paquete, tanto en almacenado en fichero como en visualización por pantalla (sólo caracteres imprimibles).

- **-h** : Muestra por pantalla la sintaxis de los diferentes parámetros de ejecución.

Para una correcta ejecución del programa, es necesario incluir las opciones '-r', '-c' y, o bien '-v' o bien '-a' o bien '-v' y '-a'. Es decir, será necesario concretar qué interfaz de red se va a utilizar para la recepción de los paquetes con '-r', se determinará cuál es el fichero de configuración inicial, y se concretarán cuales son los modos de visualización y almacenamiento con 'v' y 'a'.

10. Conclusiones

10.1. Valoración de objetivos

Se han cumplido los objetivos planteados inicialmente para el desarrollo de esta aplicación. Además, se han añadido algunas funcionalidades como por ejemplo la información de la interfaz, un filtro MAC y un filtro IP.

10.2. Propuesta de mejoras

Algunas posibles mejoras podrían ser las siguientes:

- Añadir una interfaz gráfica que facilite el uso de la aplicación.
- Posibilitar que el usuario pueda editar los ficheros de configuración y filtros dentro de la aplicación.

10.3. Sniffers

Aunque los sniffers pueden ser utilizados para diagnosticar problemas de red o estar al tanto de las acciones de los usuarios, en manos equivocadas se puede convertir en un arma peligrosa, ya que puede ser utilizado como espía robando los datos que pertenecen a otros usuarios (capturar claves, leer correos e incluso observar conversaciones en tiempo real de programas de chat como el IRC.). Para evitar que alguien pueda estar dándole un uso indebido existen varios métodos que nos permiten detectar interfaces que estén operando en modo promiscuo:

Con ifconfig es posible detectar rápidamente cualquier interfaz del host local que se encuentre en modo promiscuo.

NEPED (Network Promiscuous Ethernet Detector) es otra herramienta muy útil para detectar sniffers latentes. NEPED puede detectar actividad de sniffers en una subred.

Otra forma de aumentar la seguridad es evitar el uso de hubs, que deberán ser sustituidos por switch, dividir la red en subredes con un router, el uso de VLANS,...

Aunque puedes configurar tu LAN para hacer que los sniffers tengan mayores dificultades para entrar, estás más bien indefenso a la hora de evitar que cualquiera en Internet vea información importante. La mejor defensa en este caso es encriptar los datos, de esta forma, aunque los intercepten, no podrán entender a priori lo que dicen los paquetes. Algunas técnicas son:

SSL

"Secure Sockets Layer", SSL está disponible ampliamente en bastantes servidores y navegadores Web. Permite navegación Web encriptada, y casi siempre se usa en e-commerce cuando los usuarios introducen datos sobre tarjetas de crédito.

PGP y S/MIME

El E-mail puede ser interceptado de muchas formas. Pasa a través de firewalls, que puede que monitoricen el tráfico. A menudo queda registrado y almacenado por períodos de tiempo largos. Puede quedar mal encaminado y acabar en el buzón de otra persona. La mejor manera de mantener estos e-mails seguros es encriptándolos. Las dos maneras más comunes de hacer esto es con PGP (Pretty Good Privacy) y S/MIME (Secure MIME).

SSH

"Secure Shell", ssh se ha convertido en el standard para entrar en máquinas UNIX desde Internet. Deberíamos reemplazar el inseguro telnet por este servicio. Otros numerosos protocolos pueden usarse empleando ssh por debajo (por ejemplo, copia de ficheros.