DESARROLLO DE UN ANALIZADOR DE RED (SNIFFER)

Autor: Luis Lerones Fernández
Ingeniería Técnica Informática
Consultor: Jesús Arribi Viela
Fecha: 9 de Enero del 2006

Documento Descriptivo del proyecto

Introducción

Este proyecto se basa en el análisis y creación de un sniffer de red, una de las muchas definiciones que he encontrado acerca de lo que es un sniffer es “Un *sniffer* es un programa de para monitorear y analizar el trafico en una red de computadoras, detectando los cuellos de botellas y problemas que existan en ella”, esta sería una definición de su uso “legal”, porque también podemos interceptar las comunicaciones de los programas de conversación (MSN, IAM, ICQ…) e incluso filtrar el tráfico http, POP, SMTP y ver claves, mensajes y demás.

Pero vamos a comenzar desde el principio, estoy hablando de captura de tráfico de RED, de topologías de red, y creo que hace falta una introducción en estos aspectos para dejar claro que es el “tráfico” que se interviene y en que “topologías” de red se usan los sniffers y en cuales se dificulta su uso o se “bloquea”.
Por lo que ese será el punto inicial del proyecto, explicar que es un sniffer, y que tráfico captura y como. Aparte de comentar las características generales del TCP/IP, y de los paquetes que capturaremos posteriormente.

Objetivos

El objetivo final será el desarrollo de un sniffer propio que capturará el tráfico de la red y se podrá analizar. Para ello se generará un entorno gráfico tanto para la captura como el análisis final, obteniendo porcentaje de datos de envió y tipos de paquetes capturados.

El sniffer será un sniffer sencillo, hay que partir de la base que un sniffer se le podrían añadir miles de opciones, e incluso análisis de paquetes en busca de claves, y miles de opciones más que en un principio no tendremos en cuenta.

Planificación del proyecto

Tarea 1:

Temporización: 1 semana y media (29 de septiembre al 9 de octubre).
Descripción: Búsqueda de información para el proyecto, posibles opciones y finalmente clasificación de toda esta información:
- Temas relacionados con el *Raw Socket, formas de interceptación de información*, etc.
- Además de información relacionada con la pila TCP/IP (diferentes niveles, protocolos de trabajo…).
– Análisis de algunos sniffers actuales.
- Listar el tipo de paquetes y protocolos que usará el sniffer para su posterior análisis.
Objetivos:
- Tener claro el fin del proyecto, que tendrá que hacer el sniffer (sus funciones).
- Conocer de primera mano los tipos de respuestas y de paquetes que voy a recibir para analizarlos posteriormente.
Hitos:
- Documento descriptivo del proyecto.
- Análisis de algún sniffer para ver sus opciones y posibles mejoras. En gran medida sobre el GUI para ser más claro.

Tarea 2:

Temporización: 3 semanas (10 de octubre al 30 de octubre).
Descripción: Análisis y diseño del sniffer.
Objetivos:
- Obtener el análisis definitivo del programa a realizar...
- Dar unos primeros bocetos del GUI y realizar algún test con ellos (a nivel de usabilidad).
- Conseguir un diseño en base a lo establecido en los puntos anteriores, documentando las decisiones y dejándolo claro para su posterior programación.
Hitos:
- Análisis y Documentación de todo el sistema, además de primeros bocetos del GUI.

Tarea 3:

Temporización: 6 semanas (1 de noviembre al 11 de diciembre).

Descripción: Implementación del programa.

Objetivos:

- Desarrollo íntegro del software.
- Programar las clases que necesitaré haciendo pruebas de funcionalidad.
- Aplicar técnicas de Caja Negra para evitar errores.

Hitos:

Creación del software.

Tarea 4:

Temporización: 1 semana (12 de diciembre al 18 de diciembre).

Descripción: Juego de Pruebas, Corrección de Errores documentación de las clases.

Objetivos:

- Creación de más juegos de pruebas.
- Mejora de la documentación o creación de la misma en las clases que no la tenga.

Hitos:

Correcciones y documentación.

Tarea 5:

Temporización: 1 semana (19 de diciembre al 25 de diciembre).

Descripción: Creación del Instalador y pruebas de ejecución.

Objetivos:

- Creación del Instalador.
- Instalación en otros entornos, paso del ordenador de producción a ordenadores finales.

Hitos:

Creación del Instalador.

Tarea 6:

Temporización: 1 semana (26 de diciembre al 1 de Enero del 2006).

Descripción: Documentación.

Objetivos:

- Generar la documentación del proyecto.

Hitos:

Documentación.

Funcionamiento general de un Sniffer

Para capturar el tráfico de la red a la que estamos conectados hemos de “situar” la tarjeta de red en modo promiscuo ¿y que es esto del modo promiscuo?, por definición todos los adaptadores de red reciben los paquetes que son para ellos, (filtran por IP), pero el colocar el adaptador de red en modo promiscuo hace que no filtre, y vea todo el tráfico que está en la red.

Como acabamos de decir, un sniffer captura todos los paquetes que pasan por delante de la máquina en la que está instalado. Dicho de otra forma, un usuario que se conecte a Internet vía módem e instale un sniffer en su máquina sólo podrá capturar los paquetes de información que salgan o lleguen a su máquina.

El entorno en el que suele ser más efectivo este tipo de programas es en una Red de Área Local (LAN), montada con la topología tipo bus. En este tipo de redes todas las máquinas están conectadas a un mismo cable, que recibe el nombre de bus, y por lo tanto, todo el tráfico transmitido y recibido por todas las máquinas que pertenecen a esa red local pasa por ese cable compartido, lo que en la terminología de redes se conoce como el medio común.

El otro entorno natural de los sniffers es una máquina víctima. En este caso, es necesario tener acceso a la máquina victima para instalar el programa y el objetivo perseguido aquí es robar información que permita el acceso a otras máquinas, a las que habitualmente se accede desde esa máquina víctima.

Los sniffers funcionan por una sencilla razón: muchos de los protocolos de acceso remoto a las máquinas se transmiten las claves de acceso como texto plano, y por lo tanto, capturando la información que se transmite por la red se puede obtener este tipo de información y el acceso ilegítimo a una determinada máquina.

Como acabamos de comentar, uno de los entornos naturales para un sniffer es una LAN con topología de bus. Las redes más comunes de este tipo son las conocidas como buses Ethernet. Estas redes están formadas por una serie de máquinas, cada una de ellas equipada con una tarjeta de red Ethernet y conectadas a través de un cable coaxial, similar al utilizado por las antenas de los receptores de televisión.

En los extremos del bus es necesario situar lo que se conoce como terminadores, que no son otra cosa que una resistencia. Simplemente debemos saber que sin terminador la red no funciona.

Cada vez que una máquina de la Lan desea transmitir un dato lo hace a través de ese cable al que están conectadas todas las máquinas, por lo que todas tienen la posibilidad de ver los datos que se están transmitiendo, aunque en condiciones normales esto no sucede.

Las tarjetas de red Ethernet están construidas de tal forma que, en su modo normal de operación, sólo capturan los paquetes de datos que van dirigidos hacia ellas, ignorando la información cuyo destino es otra máquina. Lo que esto significa es que, en condiciones normales, el tráfico que circula por el bus Ethernet no puede ser capturado y es necesario activar un modo especial de funcionamiento de la tarjeta conocido como modo promiscuo.

En este modo, la tarjeta de red captura todos los paquetes que pasan por el bus en el que está pinchada y éste es el modo de operación que un sniffer necesita para llevar a cabo su finalidad.

Funcionamiento en Modo Promiscuo

Como adelantábamos en la sección anterior, cuando la tarjeta o adaptador de red se configura en modo promiscuo, captura todos los paquetes que pasan por delante de él.

La forma más inmediata de saber si un determinado adaptador de red está en un modo promiscuo es utilizar el programa ifconfig. Este programa permite configurar los adaptadores de red instalado en una determinada máquina y obtener información de esa configuración.

Cuando un adaptador de red se encuentra en modo promiscuo, ifconfig nos informa de ello. Un intruso que rompa un sistema e instale un sniffer en él sustituirá este programa por una versión modificada, de tal forma que no muestre el estado en modo promiscuo de la interfaz de red.

Para evitar esto podemos utilizar versiones del programa limpias, por ejemplo, algunas que hayamos grabado en un disco justo después de instalar el sistema, o utilizar herramientas propias para identificar este tipo de situaciones.

Sockets

Los *sockets* no son más que puntos o mecanismos de comunicación entre procesos que permiten que un proceso hable (emita o reciba información) con otro proceso incluso estando estos procesos en distintas máquinas. Esta característica de interconectividad entre máquinas hace que el concepto de socket nos sirva de gran utilidad.

Tras esta definición más formal, solo queda comentar que el uso de sockets es necesario puesto que es el método de captura de la información, para generar cabeceras, enviarlas, modificarlas… para esto necesitamos los raw sockets, que son un tipo especial que permite el acceso a protocolos de más bajo nivel como el IP, que más adelante explicaremos.

Un socket es al sistema de comunicación entre ordenadores lo que un buzón o un teléfono es al sistema de comunicación entre personas: un punto de comunicación entre dos agentes (procesos o personas respectivamente) por el cual se puede emitir o recibir información. La forma de referenciar un socket por los procesos implicados es mediante un descriptor del mismo tipo que el utilizado para referenciar ficheros. Debido a esta característica, se podrá realizar redirecciones de los archivos de E/S estándar (descriptores 0,1 y 2) a los sockets y así combinar entre ellos aplicaciones de la red. Todo nuevo proceso creado heredará, por tanto, los descriptores de sockets de su padre.

La comunicación entre procesos a través de sockets se basa en la filosofía CLIENTE-SERVIDOR: un proceso en esta comunicación actuará de proceso servidor creando un socket cuyo nombre conocerá el proceso cliente, el cual podrá "hablar" con el proceso servidor a través de la conexión con dicho socket nombrado.

El proceso crea un socket sin nombre cuyo valor de vuelta es un descriptor sobre el que se leerá o escribirá, permitiéndose una comunicación bidireccional, característica propia de los sockets y que los diferencie de los pipes, o canales de comunicación unidireccional entre procesos de una misma máquina. El mecanismo de comunicación vía sockets tiene los siguientes pasos:

1º) El proceso servidor crea un socket con nombre y espera la conexión.

2º) El proceso cliente crea un socket sin nombre.
3º) El proceso cliente realiza una petición de conexión al socket servidor.
4º) El cliente realiza la conexión a través de su socket mientras el proceso servidor mantiene el socket servidor original con nombre.

Es muy común en este tipo de comunicación lanzar un proceso hijo, una vez realizada la conexión, que se ocupe del intercambio de información con el proceso cliente mientras el proceso padre servidor sigue aceptando conexiones. Para eliminar esta característica se cerrará el descriptor del socket servidor con nombre en cuanto realice una conexión con un proceso socket cliente.

-> Todo socket viene definido por dos características fundamentales:

- El **tipo** del socket, que indica la naturaleza del mismo, el tipo de comunicación que puede generarse entre los sockets.

- El **dominio** del socket especifica el conjunto de sockets que pueden establecer una comunicación con el mismo.

Tipos de Sockets

Define las propiedades de las comunicaciones en las que se ve envuelto un socket, esto es, el tipo de comunicación que se puede dar entre cliente y servidor.
Estas pueden ser:

- Fiabilidad de transmisión.
- Mantenimiento del orden de los datos.
- No duplicación de los datos.
- El "Modo Conectado" en la comunicación.
- Envío de mensajes urgentes.

Los tipos disponibles son los siguientes:

* Tipo **SOCK_DGRAM**: sockets para comunicaciones en modo no conectado, con envío de datagramas de tamaño limitado (tipo telegrama). En dominios

Internet como la que nos ocupa el protocolo del nivel de transporte sobre el que se basa es el **UDP**.

* Tipo **SOCK_STREAM**: para comunicaciones fiables en modo conectado, de dos vías y con tamaño variable de los mensajes de datos. Por debajo, en dominios Internet, subyace el protocolo TCP.

* Tipo **SOCK_RAW**: permite el acceso a protocolos de más bajo nivel como el IP (nivel de red)

* Tipo **SOCK_SEQPACKET**: tiene las características del SOCK_STREAM pero además el tamaño de los mensajes es fijo.

Tráfico capturado

Desde el principio venimos hablando de tráfico, captura de tráfico, pero ¿Qué es este tráfico? Hemos explicado como se captura, pero no que es en si.
En este apartado vamos a dar respuesta a estas preguntas. Daremos un vistazo general sobre el protocolo TCP/IP, y comentaremos los diferentes tipos de paquetes que vamos a capturar, analizándolos pero sin enterar a ellos en profundidad, puesto que suponemos que la persona que quiera usar un sniffer ha de tener unos conocimientos medios sobre redes y protocolos.

El Protocolo TCP/IP

Una red es una configuración de computadora que intercambia información. Pueden proceder de una variedad de fabricantes y es probable que tenga diferencias tanto en hardware como en software, para posibilitar la comunicación entre estas es necesario un conjunto de reglas formales para su interacción. A estas reglas se les denominan protocolos.

Un protocolo es un conjunto de reglas establecidas entre dos dispositivos para permitir la comunicación entre ambos. Internet no es dependiente de la máquina ni

del sistema operativo utilizado. De esta manera, podemos transmitir información entre un servidor Unix y un ordenador que utilice Windows 98. O entre plataformas completamente distintas como Macintosh, Alpha o Intel. Es más: entre una máquina y otra generalmente existirán redes distintas: redes Ethernet, redes Token Ring e incluso enlaces vía satélite. Como vemos, está claro que no podemos utilizar ningún protocolo que dependa de una arquitectura en particular. Lo que estamos buscando es un método de interconexión general que sea válido para cualquier plataforma, sistema operativo y tipo de red. La familia de protocolos que se eligieron para permitir que Internet sea una *Red de redes* es TCP/IP. Nótese aquí que hablamos de familia de protocolos ya que son muchos los protocolos que la integran, aunque en ocasiones para simplificar hablemos sencillamente del protocolo TCP/IP.

El protocolo TCP/IP tiene que estar a un nivel superior del tipo de red empleado (y más adelante comentaremos los tipos de red existente) y funcionar de forma transparente en cualquier tipo de red. Y a un nivel inferior de los programas de aplicación (páginas WEB, correo electrónico…) particulares de cada sistema operativo. Todo esto nos sugiere el siguiente modelo de referencia:

Capa de aplicación (HTTP, SMTP, FTP, TELNET…)
Capa de transporte (UDP, TCP)
Capa de red (IP)
Capa de acceso a la red (Ethernet, Token Ring…)
Capa física (cable coaxial, par trenzado…)

El nivel más bajo es la capa física. Aquí nos referimos al medio físico por el cual se transmite la información. Generalmente será un cable aunque no se descarta cualquier otro medio de transmisión como ondas o enlaces vía satélite.

La capa de acceso a la red determina la manera en que las estaciones (ordenadores) envían y reciben la información a través del soporte físico proporcionado por la capa anterior. Es decir, una vez que tenemos un cable, ¿cómo

se transmite la información por ese cable? ¿Cuándo puede una estación transmitir? ¿Tiene que esperar algún turno o transmite sin más? ¿Cómo sabe una estación que un mensaje es para ella? Pues bien, son todas estas cuestiones las que resuelve esta capa.

Las dos capas anteriores quedan a un nivel inferior del protocolo TCP/IP, es decir, no forman parte de este protocolo. La capa de red define la forma en que un mensaje se transmite a través de distintos tipos de redes hasta llegar a su destino. El principal protocolo de esta capa es el IP aunque también se encuentran a este nivel los protocolos ARP, ICMP e IGMP. Esta capa proporciona el direccionamiento IP y determina la ruta óptima a través de los encaminadores (*routers*) que debe seguir un paquete desde el origen al destino.

La capa de transporte (protocolos TCP y UDP) ya no se preocupa de la ruta que siguen los mensajes hasta llegar a su destino. Sencillamente, considera que la comunicación extremo a extremo está establecida y la utiliza. Además añade la noción de puertos, como veremos más adelante.

Una vez que tenemos establecida la comunicación desde el origen al destino nos queda lo más importante, ¿qué podemos transmitir? La capa de aplicación nos proporciona los distintos servicios de Internet: correo electrónico, páginas Web, FTP, TELNET…

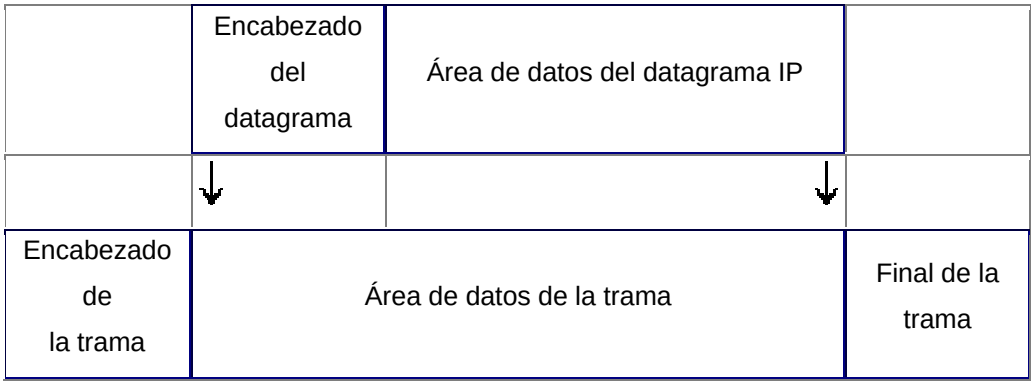
Protocolo IP

IP es el principal protocolo de la capa de red. Este protocolo define la unidad básica de transferencia de datos entre el origen y el destino, atravesando toda la red de redes. Además, el software IP es el encargado de elegir la ruta más adecuada por la que los datos serán enviados. Se trata de un sistema de entrega de paquetes (llamados *datagramas IP*) que tiene las siguientes características:

- Es no orientado a conexión debido a que cada uno de los paquetes puede seguir rutas distintas entre el origen y el destino. Entonces pueden llegar duplicados o desordenados.
- Es no fiable porque los paquetes pueden perderse, dañarse o llegar retrasados.

Formato del datagrama IP

El datagrama IP es la unidad básica de transferencia de datos entre el origen y el destino. Viaja en el campo de datos de las tramas físicas de las distintas redes que va atravesando. Cada vez que un datagrama tiene que atravesar un router, el datagrama *saldrá* de la trama física de la red que abandona y se *acomodará* en el campo de datos de una trama física de la siguiente red. Este mecanismo permite que un mismo datagrama IP pueda atravesar redes distintas: enlaces punto a punto, redes ATM, redes Ethernet, redes Token Ring, etc. El propio datagrama IP tiene también un campo de datos; será aquí donde viajen los paquetes de las capas superiores.



0										10										20										30	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	3	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
VERS				HLEN				Tipo de servicio								Longitud total															
Identificación																				Banderas				Desplazamiento de fragmento							
TTL								Protocolo								CRC cabecera															
Dirección IP origen																															
Dirección IP destino																															
Opciones IP (si las hay)																												Relleno			
Datos																															
...																															

Campos del datagrama IP:

- **VERS** (4 bits). Indica la versión del protocolo IP que se utilizó para crear el datagrama. Actualmente se utiliza la versión 4 (IPv4) aunque ya se están preparando las especificaciones de la siguiente versión, la 6 (IPv6).
- **HLEN** (4 bits). Longitud de la cabecera expresada en múltiplos de 32 bits. El valor mínimo es 5, correspondiente a 160 bits = 20 bytes.
- **Tipo de servicio** *(Type Of Service)*. Los 8 bits de este campo se dividen a su vez en:
    - **Prioridad** (3 bits). Un valor de 0 indica baja prioridad y un valor de 7, prioridad
      máxima.
    - Los siguientes tres bits indican cómo se prefiere que se transmita el mensaje, es decir, son sugerencias a los encaminadores que se encuentren a su paso los cuales pueden tenerlas en cuenta o no.
    - **Bit D** *(Delay)*. Solicita retardos cortos (enviar rápido).
    - **Bit T** *(Throughput)*. Solicita un alto rendimiento (enviar mucho en el menor tiempo posible).
    - **Bit R** *(Reliability)*. Solicita que se minimice la probabilidad de que el datagrama se pierda o resulte dañado (enviar bien).
    - Los siguientes dos bits no tienen uso.
- **Longitud total** (16 bits). Indica la longitud total del datagrama expresada en bytes. Como el campo tiene 16 bits, la máxima longitud posible de un datagrama será de 65535 bytes.
- **Identificación** (16 bits). Número de secuencia que junto a la dirección origen, dirección destino y el protocolo utilizado identifica de manera única un datagrama en toda la red. Si se trata de un datagrama fragmentado, llevará la misma identificación que el resto de fragmentos.
- **Banderas** o indicadores (3 bits). Sólo 2 bits de los 3 bits disponibles están actualmente utilizados. El bit de *Más fragmentos* (**MF**) indica que no es el último datagrama. Y el bit de *No fragmentar* **(NF)** prohíbe la fragmentación del datagrama. Si este bit está activado y en una determinada red se requiere fragmentar el datagrama, éste no se podrá transmitir y se descartará.
- **Desplazamiento de fragmentación** (13 bits). Indica el lugar en el cual se insertará el fragmento actual dentro del datagrama completo, medido en unidades de 64 bits. Por esta razón los campos de datos de todos los fragmentos menos el último tienen una longitud múltiplo de 64 bits. Si el paquete no está fragmentado, este campo tiene el valor de cero.

- **Tiempo de vida** o TTL (8 bits). Número máximo de segundos que puede estar un datagrama en la red de redes. Cada vez que el datagrama atraviesa un router se resta 1 a este número. Cuando llegue a cero, el datagrama se descarta y se devuelve un mensaje ICMP de tipo "tiempo excedido" para informar al origen de la incidencia.
- **Protocolo** (8 bits). Indica el protocolo utilizado en el campo de datos: 1 para ICMP, 2 para IGMP, 6 para TCP y 17 para UDP.
- **CRC cabecera** (16 bits). Contiene la suma de comprobación de errores sólo para la cabecera del datagrama. La verificación de errores de los datos corresponde a las capas superiores.
- **Dirección origen** (32 bits). Contiene la dirección IP del origen.
- **Dirección destino** (32 bits). Contiene la dirección IP del destino.
- **Opciones IP**. Este campo no es obligatorio y especifica las distintas opciones solicitadas por el usuario que envía los datos (generalmente para pruebas de red y depuración).
- **Relleno.** Si las opciones IP (en caso de existir) no ocupan un múltiplo de 32 bits, se completa con bits adicionales hasta alcanzar el siguiente múltiplo de 32 bits (recuérdese que la longitud de la cabecera tiene que ser múltiplo de 32 bits).

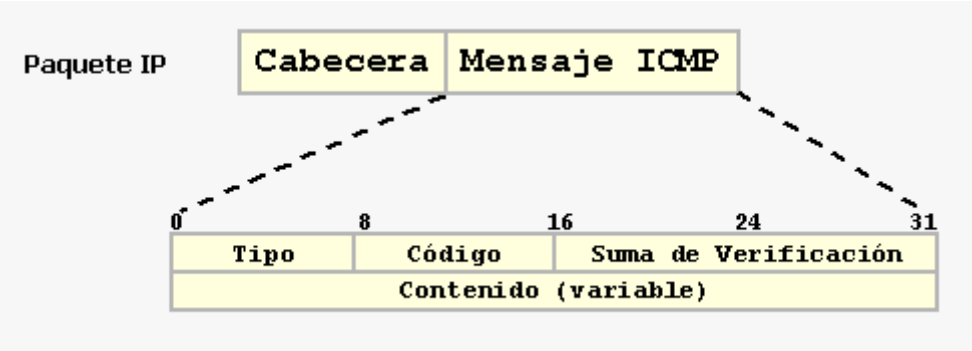
Protocolos que Capturaremos

ICMP (Internet Control Message Protocol)

Permite el intercambio de mensajes de control y de supervisión entre dos ordenadores. Toda anomalía detectada por el protocolo IP provoca el intercambio de mensajes ICMP entre los nodos de la red. Este protocolo forma parte de la capa internet y usa la facilidad de enviar paquetes IP para enviar mensajes.

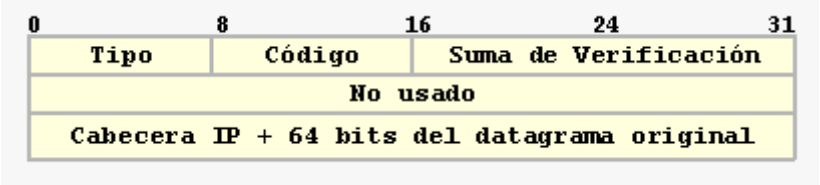
ICMP es un protocolo de control que utilizan los dispositivos de encaminamiento para notificar las diferentes incidencias que puede haber en una red IP. Está definido en el RFC 792. Proporciona información de realimentación sobre los problemas que ocurren y por ejemplo se utiliza cuando:

- Un datagrama no puede alcanzar su destino
- El dispositivo de encaminamiento no tiene la capacidad de almacenar temporalmente el datagrama para poderlo reenviar, y
- El dispositivo de encaminamiento indica a un ordenador que envíe el tráfico por una ruta mas corta (redireccionamiento de rutas).Cada mensaje ICMP se encapsula en un paquete IP y luego es enviado de la forma habitual. Como los mensajes ICMP se transmiten en mensajes IP, no se puede garantizar que llegue a su destino.



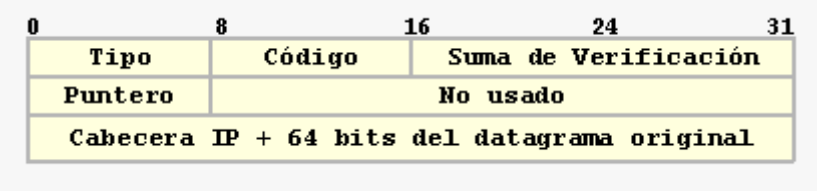
A continuación se describen los diferentes tipos de mensajes ICMP que hay. En las figuras puede verse el formato correspondiente a cada uno de ellos. El número de campos del mensaje, la interpretación de cada campo y la cantidad de datos depende del tipo de mensaje.

- **ICMP (3) - Detectar destinos inalcanzables**: Cuando un dispositivo de encaminamiento detecta que la red o el servidor de destino es inalcanzable, envía un mensaje de destino inalcanzable al emisor del datagrama. El ordenador destino también puede enviar este mensaje si el protocolo de usuario o algún punto de acceso al servicio de nivel superior no es alcanzable. Por último si en algún dispositivo de encaminamiento es necesario fragmentar un datagrama pero está activo el indicador de no fragmentación, también se envía un mensaje de destino inalcanzable.



- **ICMP (11) - Tiempo excedido**: Será devuelto por un dispositivo de encaminamiento intermedio si ha expirado el tiempo de vida del datagrama o si no ha podido reensamblar en el paquete en un tiempo determinado. Este mensaje es un síntoma de que los paquetes están en un ciclo, de que hay mucha congestión en la red o de que los valores de temporización son demasiado bajos. Este paquete contiene la cabecera del paquete IP que ha sido eliminado y los primeros 8 bytes de su contenido.

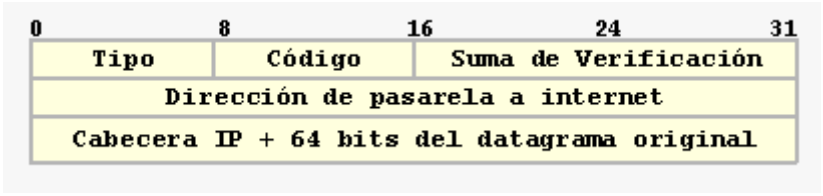
- **ICMP (12) - Problema de parámetros**: Este mensaje es enviado cuando hay algún error sintáctico o semántico en la cabecera IP. Por ejemplo, un argumento incorrecto en una opción. El campo parámetro de la respuesta contiene un puntero al byte de la cabecera original donde se detectó el error. Este mensaje indica un fallo en el programa IP del emisor o en alguno de los dispositivos de encaminamiento.



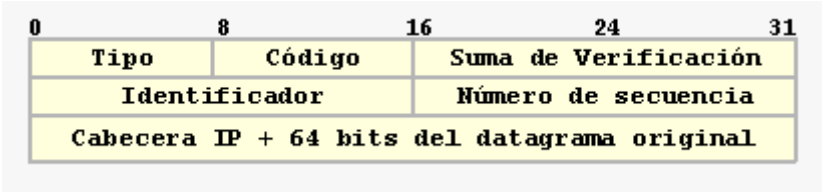
0	8	16	24	31
Tipo	Código	Suma de Verificación		
Puntero	No usado			
Cabecera IP + 64 bits del datagrama original				

- **ICMP (4) – Petición de control de flujo** (ralentización del origen): Cuando los datagramas llegan demasiado deprisa para ser procesados, el servidor de destino o un dispositivo de encaminamiento intermedio envía un mensaje al emisor indicándole que ralentice o pare temporalmente el envío de datagramas. Este mecanismo puede activarse antes de que se produzca la congestión y en ese caso el mensaje original si que se habrá recibido correctamente. En la actualidad se utiliza poco pues, cuando hay congestión, esos mensajes lo único que hacen es empeorar la situación. Actualmente el control de congestión se realiza en la capa de transporte como veremos mas adelante.

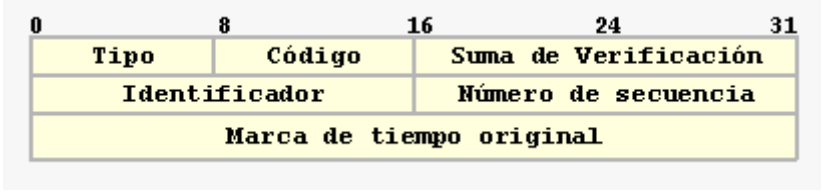
- **ICMP (5) - Redireccionando rutas**: Un dispositivo de encaminamiento envía un mensaje ICMP de redirección para decirle al emisor que use otro dispositivo de encaminamiento, presumiblemente porque es una mejor elección. Este mensaje sólo puede ser usado cuando el ordenador emisor está en la misma red que los dos dispositivos de encaminamiento.

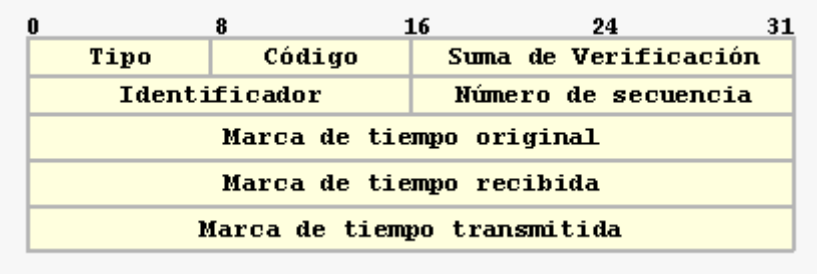


- **ICMP (0) - Eco** e **ICMP (8) - respuesta a eco**: permiten comprobar que el protocolo Internet de un sistema remoto está funcionando, es decir, si un ordenador es alcanzable y esta vivo. El receptor de un mensaje eco está obligado a responder con un mensaje respuesta a eco. El mensaje de eco tiene asociado un identificador y un número de secuencia que coinciden con los que hay en el mensaje de respuesta a eco. Como en los sistemas multitarea puede haber varias peticiones en curso es necesario utilizar un identificador de secuencia.

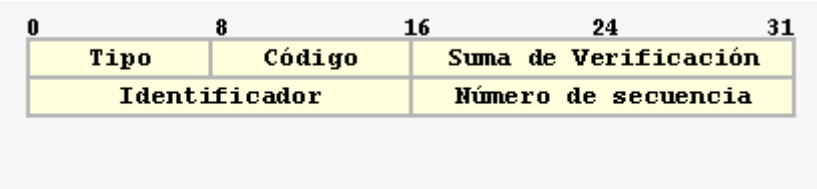


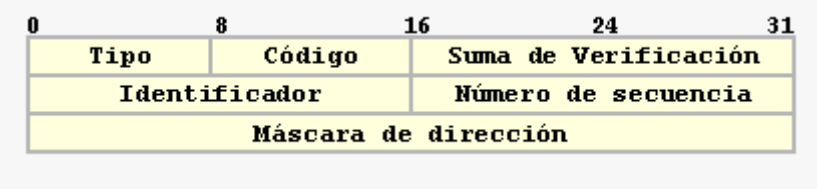
- **ICMP (13) - Marca de tiempo** y **ICMP (14) - respuesta a la marca de tiempo** proporcionan un mecanismo para saber el retardo que hay. El emisor pone una marca de tiempo para indicar cuando envía el mensaje y el receptor indica cuando recibió el mensaje y cuando lo ha reenviado. Si el mensaje de marca de tiempo se envía usando un encaminamiento por la fuente destino, podemos determinar las características del retardo de una ruta particular.





- **ICMP (17) - Petición de máscara de dirección** y **ICMP (18) - respuesta a la máscara de dirección** se utilizan en los entornos que incluyen subredes.





- **ICMP (15) – Petición de información y ICMP (16) – respuesta de información.**

- **ICMP (9) – Petición de rutas y ICMP (10) – publicación de rutas.**

Protocolo UDP

Este protocolo es no orientado a la conexión, y por lo tanto no proporciona ningún tipo de control de errores ni de flujo, aunque si que utiliza mecanismos de detección de errores. Cuando se detecta un error en un datagrama en lugar de entregarlo a la aplicación se descarta.

Este protocolo se ha definido teniendo en cuenta que el protocolo del nivel inferior (el protocolo IP) también es no orientado a la conexión y puede ser interesante tener un protocolo de transporte que explote estas características.

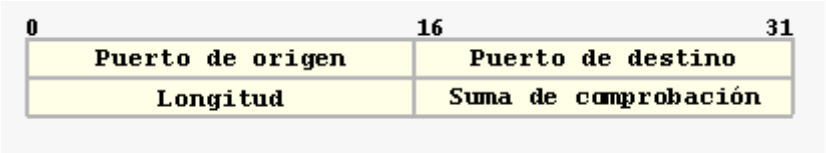
Como el protocolo es no orientado a la conexión cada datagrama UDP existe independientemente del resto de datagramas UDP.

El protocolo UDP es muy sencillo y tiene utilidad para las aplicaciones que requieren pocos retardos o para ser utilizado en sistemas sencillos que no pueden implementar el protocolo TCP.

Las características del protocolo UDP son:

- No garantiza la fiabilidad. No podemos asegurar que cada datagrama UDP transmitido llegue a su destino. Es un protocolo del tipo best-effort porque hace lo que puede para transmitir los datagramas hacia la aplicación pero no puede garantizar que la aplicación los reciban.
- No preserva la secuencia de la información que proporciona la aplicación. La información se puede recibir desordenada (como ocurría en IP) y la aplicación debe estar preparada por si se pierden datagramas, llegan con retardo o llegan desordenados.

La siguiente figura muestra los campos de un datagrama UDP y como se forma el datagrama IP.



Un datagrama consta de una cabecera y de un cuerpo en el que se encapsulan los datos. La cabecera consta de los siguientes campos:

- Los campos puerto origen y puerto destino son de 16 bits e identifican las aplicaciones en la máquina origen y en la máquina destino.

- El campo longitud es de 16 bits e indica en bytes la longitud del datagrama UDP incluyendo la cabecera UDP. En realidad es la longitud del datagrama IP menos el tamaño de la cabecera IP. Como la longitud máxima del datagrama IP es de 65.535 bytes y la cabecera estándar de IP es de 20 bytes, la longitud máxima de un datagrama UDP es de 65.515 bytes.

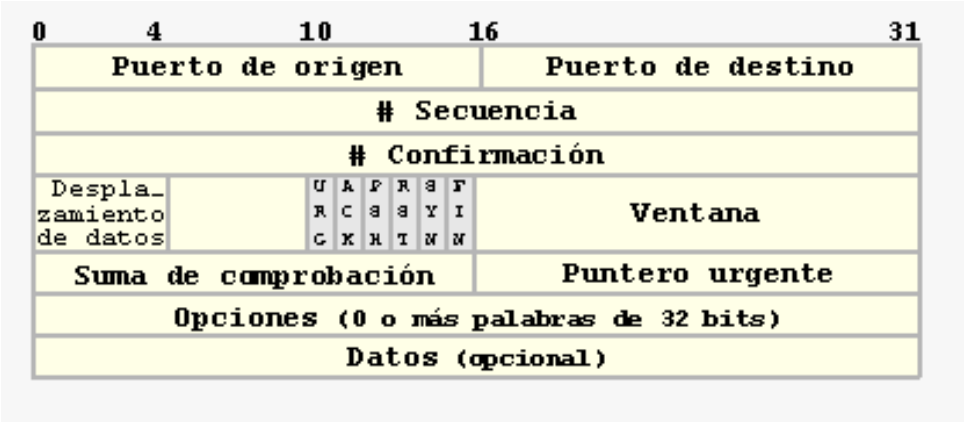
- El campo suma de comprobación (checksum) es un campo opcional de 16 bits que, a diferencia del campo equivalente de la cabecera IP que solo protegía la cabecera, protege tanto la cabecera como los datos.

Como el protocolo UDP no está orientado a la conexión y no envía ningún mensaje para confirmar que se han recibido los datagramas, su utilización es adecuada cuando queremos transmitir información en modo multicast (a muchos destinos) o en modo broadcast (a todos los destinos) pues no tiene sentido esperar la confirmación de todos los destinos para continuar con la transmisión. También es importante tener en cuenta que si en una transmisión de este tipo los destinos enviarán confirmación, fácilmente el emisor se vería colapsado, pues por cada paquete que envía recibiría tantas confirmaciones como destinos hayan recibido el paquete.

Lo que realmente proporciona UDP respecto a IP es la posibilidad de multiplexación de aplicaciones. La dirección del puerto permite identificar aplicaciones gracias a la dirección del puerto.

Protocolo TCP

La unidad de datos de este protocolo recibe el nombre de segmento TCP. Como la cabecera debe implementar todos los mecanismos del protocolo su tamaño es bastante grande, como mínimo 20 bytes.
En la siguiente figura se puede ver el formato de la cabecera TCP.



A continuación hay una descripción de cada uno de los campos que forman la cabecera:

- **Puerto origen** (16 bits): Es el punto de acceso de la aplicación en el origen.

- **Puerto destino** (16 bits): Es el punto de acceso de la aplicación en el destino.

- **Número de secuencia** (32 bits): Identifica el primer byte del campo de datos. En este protocolo no se enumeran segmentos sino bytes, por lo que este número indica el primer byte de datos que hay en el segmento. Al principio de la conexión se asigna un número de secuencia inicial (ISN, Initial Sequence Number) y a continuación los bytes son numerados consecutivamente.

- **Número de confirmación (ACK)** (32 bits): El protocolo TCP utiliza la técnica de piggybacking para reconocer los datos. Cuando el bit ACK está activo, este campo contiene el número de secuencia del primer byte que espera recibir. Dicho de otra manera, el número ACK - 1 indica el último bit reconocido.

- **Longitud de la cabecera** (4 bits): Indica el número de palabras de 32 bits que hay en la cabecera. De esta manera el TCP puede saber donde se acaba la cabecera y por lo tanto donde empieza los datos. Normalmente el tamaño de la cabecera es de 20 bytes por lo que en este campo se almacenará el número 5. Si el TCP utiliza

todos los campos de opciones la cabecera puede tener una longitud máxima de 60 bytes almacenándose en este campo el valor 15.

- **Reservado** (6 bits): Se ha reservado para su uso futuro y se inicializa con ceros.

- **Indicadores o campo de control** (6 bits): Cada uno de los bits recibe el nombre de indicador y cuando está a 1 indica una función específica del protocolo.

- **URG:** Hay datos urgentes y en el campo "puntero urgente" se indica el número de datos urgentes que hay en el segmento.

- **ACK:** Indica que tiene significado el número que hay almacenado en el campo "número de confirmación".

- **PSH:** Sirve para invocar la función de carga (push). Como se ha comentado anteriormente con esta función se indica al receptor que debe pasar a la aplicación todos los datos que tenga en la memoria intermedia sin esperar a que sean completados. De esta manera se consigue que los datos no esperen en la memoria receptora hasta completar un segmento de dimensión máxima. No se debe confundir con el indicador URG que sirve para señalar que la aplicación ha determinado una parte del segmento como urgente.

- **RST:** Sirve para hacer un reset de la conexión.

- **SYN:** Sirve para sincronizar los números de secuencia.

- **FIN:** Sirve para indicar que el emisor no tiene mas datos para enviar.

- **Ventana** (16 bits): Indica cuantos bytes tiene la ventana de transmisión del protocolo de control de flujo utilizando el mecanismo de ventanas deslizantes. A diferencia de lo que ocurre en los protocolos del nivel de enlace, en los que la ventana era constante y contaba tramas, en el TCP la ventana es variable y cuenta bytes. Contiene el número de bytes de datos comenzando con el que se indica en el campo de confirmación y que el que envía está dispuesto a aceptar.

- **Suma de comprobación** (16 bits): Este campo se utiliza para detectar errores mediante el complemento a uno de la suma en módulo 2^{16} - 1 de todas las palabras de 16 bits que hay en el segmento mas una pseudo-cabecera que se puede ver en la siguiente figura. La pseudo-cabecera incluye los siguientes campos de la

cabecera IP: dirección internet origen, dirección internet destino, el protocolo y un campo longitud del segmento. Con la inclusión de esta pseudo-cabecera, TCP se protege a si mismo de una transmisión errónea de IP. Esto es, si IP lleva un segmento a un computador erróneo, aunque el segmento esté libre de errores, el receptor detectará el error de transmisión (INCLUIR FIG. 17.15)

- **Puntero urgente** (16 bits): Cuando el indicador URG está activo, este campo indica cual es el último byte de datos que es urgente. De esta manera el receptor puede saber cuantos datos urgentes llegan. Este campo es utilizado por algunas aplicaciones como telnet, rlogin y ftp.

- **Opciones** (variable): Si está presente permite añadir una única opción de entre las siguientes:

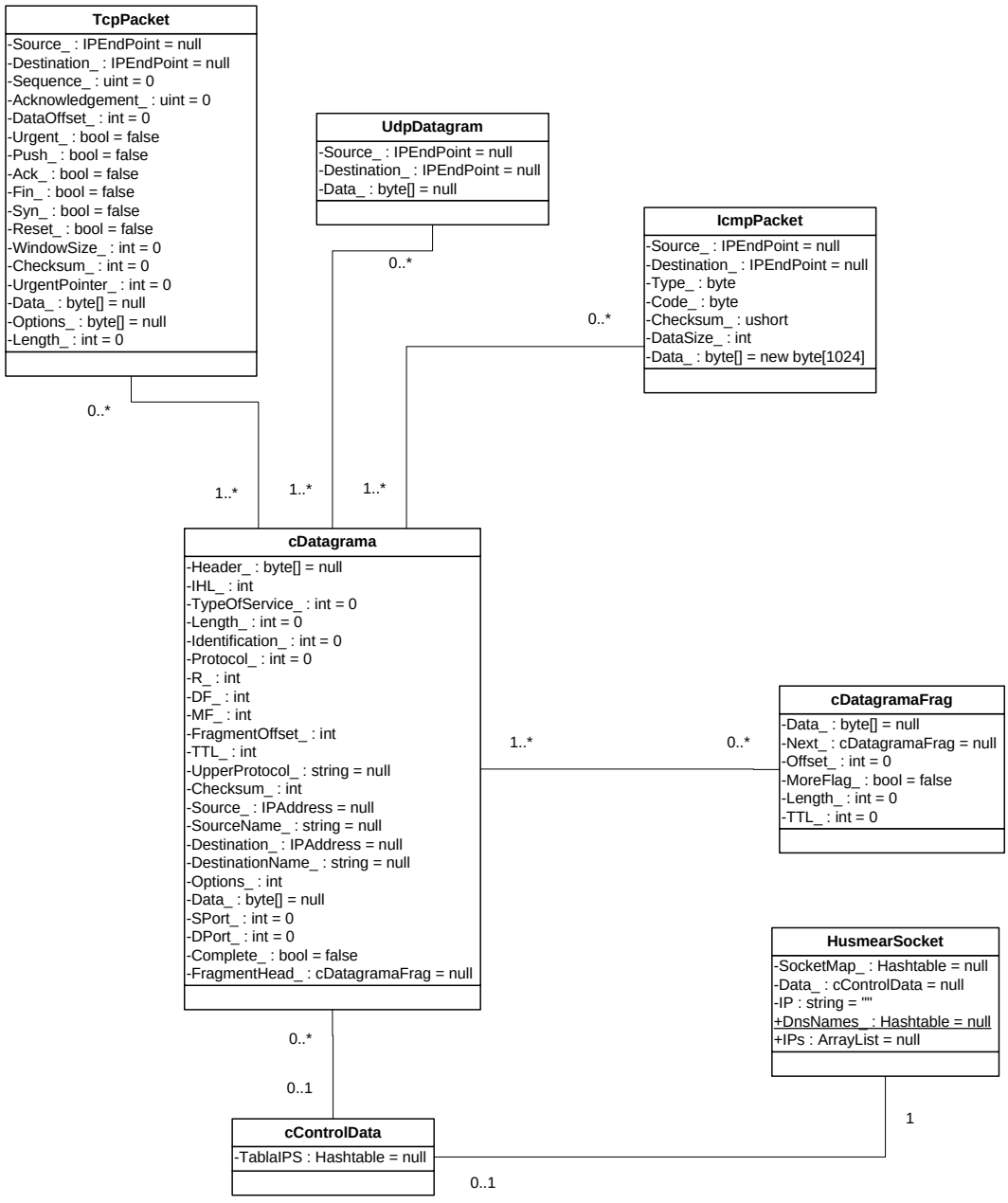
- **Timestamp** para marcar en que momento se transmitió el segmento y de esta manera monitorizar los retardos que experimentan los segmentos desde el origen hasta el destino.

- **Aumentar el tamaño de la ventana.**

- **Indicar el tamaño máximo del segmento** que el origen puede enviar.
Como TCP ha sido diseñado para trabajar con IP, algunos parámetros de usuario se pasan a través de TCP a IP para su inclusión en la cabecera IP. Por ejemplo: prioridad (campo de 3 bits), retardo-normal/ retardo- bajo, rendimiento-normal/rendimiento-alto, seguridad-normal/seguridad-alta y protección (campo de 11 bits)

Fase de Análisis y Diseño

Diagrama de Clases



En este diagrama está expresada la lógica del programa, no aparecen las funciones, aunque si los atributos que tienen. Como elemento central pongo la clase cControlData, puesto que esta clase es la que tiene la tabla de Paquetes Capturados, y solo hay una tabla, dicha tabla contiene muchos datagramas, los cuales están formados por paquetes de los diferentes tipos, y cuando es necesario

un propio paquete está compuesto por varios fragmentos de un mismo paquete. Y finalmente la clase cControlData está relacionada con la clave HusmearSocket que es la que crea el objeto socket que está escuchando, en este caso se crea uno, aunque se le pueden añadir Ips a las que ha de husmear.

Obtener el análisis definitivo del programa a realizar.

El programa finalmente capturará los paquetes de los siguientes tipos: TCP, UDP, ICMP. De dichos paquetes la información que mostrará será:
El Número que tienen, el Origen, la IP, el Destino, la IP, el puerto de destino, el Protocolo.

El resultado final lo he obtenido de un análisis de otros softwares, para ver cual es la información básica de todo sniffer (considero que en el proyecto se basa el crear un “modelo” que posteriormente se podría ampliar para obtener así un sniffer mucho más completo y complejo, que realizara cualquier captura, y analizara en profundidad todos los paquetes. Esta información la expondré de mejor manera al final del documento en el apartado de “Futuras Versiones”.

Aquí veremos unos esquemas UML de la aplicación, en el que explicaremos las decisiones tomadas, veremos las clases usadas, y expondremos la técnica de programación usada, hay que tener en cuenta que el desarrollo se realizará en .NET v1.1 (actualmente está en el mercado la versión .NET 2.0 con muchas mejoras, pero el desarrollo lo comencé con la versión 1.1 y para evitar incompatibilidades y posibles problemas el desarrollo definitivo está realizado con la versión 1.1).

También en los próximos puntos se verá como queda el diseño, y por los puntos que ha pasado el GUI, para obtener finalmente una interfaz que sea fácil, e intuitiva. Para esto hemos utilizado las técnicas de Usabilidad expuestas en la asignatura de Interacción Humana con los Computadores (IHO). Para dejar claro que es IHO podemos dar alguna definición como puede ser: "El estudio de la gente, la tecnología y las formas como se influencian. Estudiamos IHO para determinar cómo podemos hacer la tecnología más usable para las personas."

Pero ¿Por qué uso las técnicas de IHO para el proyecto? ¿Qué beneficios aporta al diseño? Todas estas preguntas y muchas más se pueden formular, y en ocasiones

se puede pensar en que la aplicación de estas técnicas a un diseño ofrece más una perdida de tiempo que un beneficio, siendo mentira.

Vivimos en un mundo tecnológico repleto de aparatos complicados que, en muchos casos, parecen más diseñados para dificultarnos la vida que para facilitárnosla. En un mundo de manuales de instrucciones, en el que sin ellos nos puede llegar a resultar imposible realizar una tarea, por ejemplo programar un video sin el manual… Dadas estas premisas, no parece extraño que cada vez más la IHO adquiera una creciente importancia y nos ayude a lograr un mundo sin manuales de instrucciones, que para nosotros sea definitivamente un mundo más feliz.

Habitualmente somos nosotros los que debemos adaptarnos al objeto con el que queremos interactuar (cuando hablamos de objeto es para hacerlo más Standard la afirmación, puesto que la IHO se aplica a todo aquello que necesite una atención por nuestra parte). La IHO satisfará su objetivo último en la medida en que consiga que los ordenadores, digitales, exactos y deterministas, se adapten a las personas, analógicas, inexactas y no deterministas, y no al revés.

Por eso, los sistemas diseñados tendrán que ser:

- Usables, intuitivos (fáciles de aprender y de utilizar).
- Seguros (que no perjudiquen el usuario ni a corto ni a largo plazo)
- Funcionales (que satisfagan las necesidades de los usuarios)
- Eficientes (que lo que hagan lo hagan de la mejor manera posible de acuerdo con unos criterios determinados).

Aplicación de las Técnicas de IHO al Interfaz

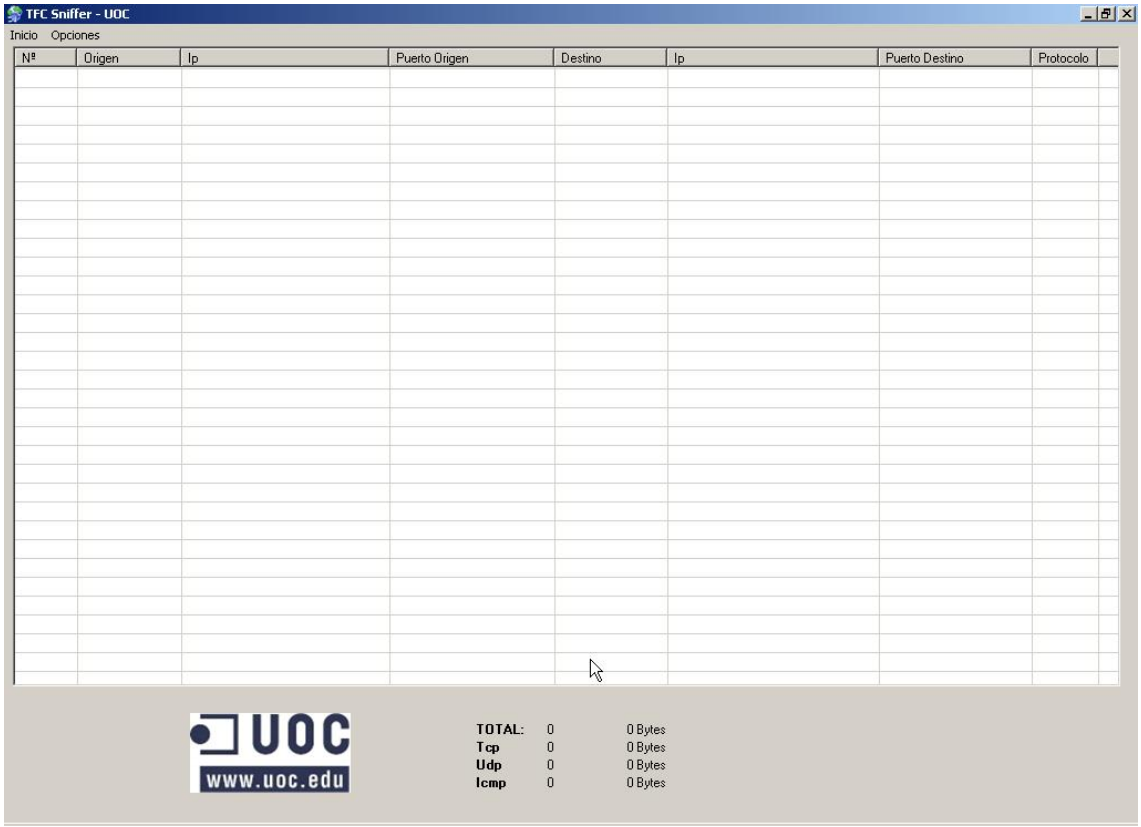
Para realizar el análisis del Interfaz, debemos de analizar opción a opción, en este caso nos resultó muy sencillo dada su simplicidad (tras varias aplicaciones anteriores).
La idea es a cada tarea realizar unas preguntas:

- **P1: ¿Está la acción correcta disponible en el interfaz?**
- **P2: ¿Cómo de fácil se puede relacionar la descripción de la acción (icono, etiqueta, nombre del comando,…) con nuestro objetivo?**
- **P3: ¿La respuesta del sistema a la acción seleccionada muestra progreso hacia el objetivo final del usuario?**

Con estas respuestas podremos obtener los golfos de evaluación y con ellos ver donde hemos de mejorar.
La idea es dejar finalmente (como creo que ha quedado) un interfaz claro y sencillo.

También se pueden usar Test con personas, esto sería darles el programa a múltiples personas con roles diferentes y pedirles a todos que realicen las mismas tareas, controlando el tiempo y los movimientos que han de hacer hasta conseguirlo. Por las condiciones de este programa supongo un nivel medio-alto para los usuarios que lo deban de usar, por lo que preferí la realización del primer test en el que yo mismo puedo ver donde fallaba y arreglarlo.

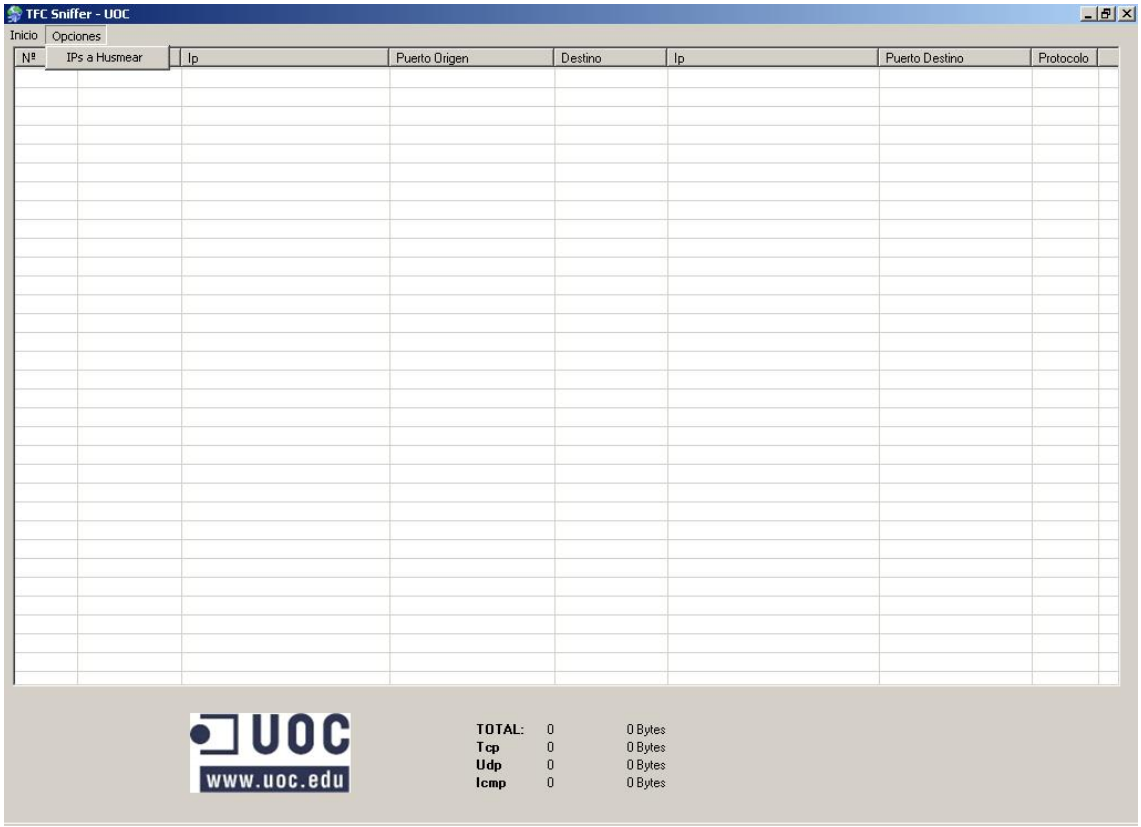
GUI Final del Programa

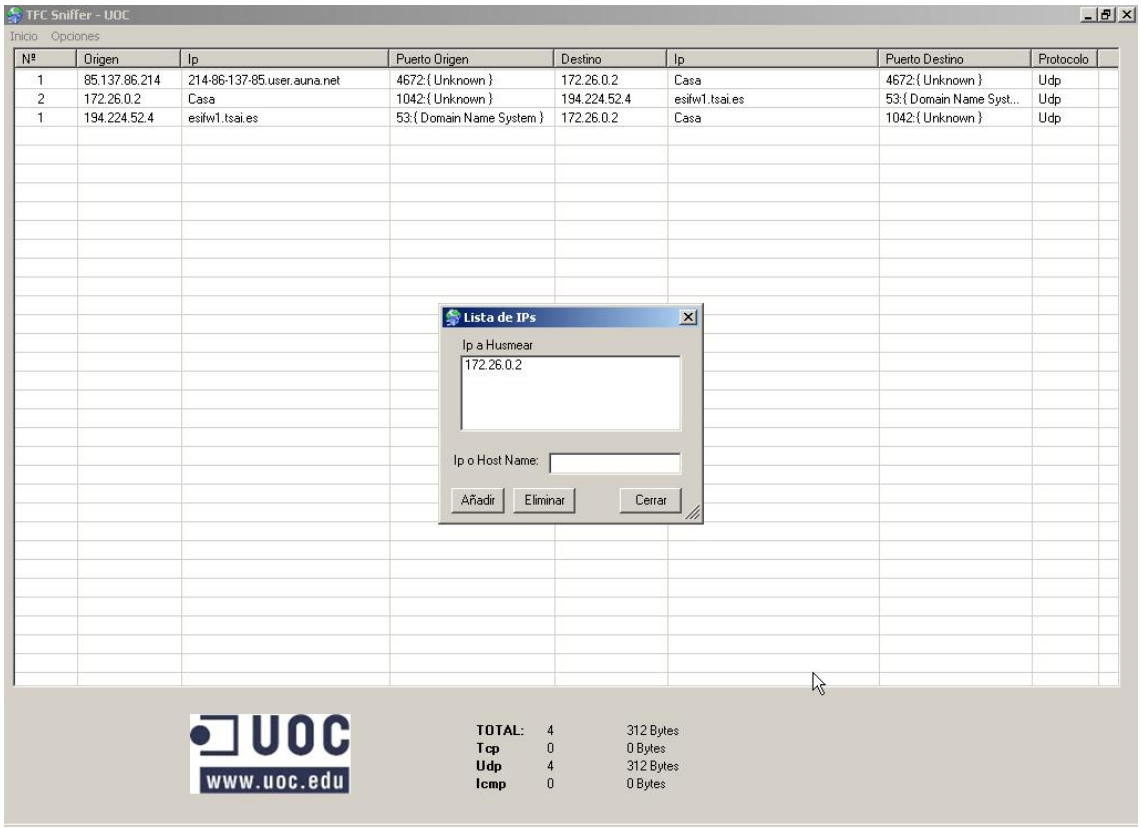


En este primer pantallaza podemos observar de forma general el estilo que ha tomado el programa tras aplicarle las técnicas de IHO en cuanto a la evaluación del GUI, y la eliminación de los Golfos innecesarios. Finalmente ha quedado un programa muy sencillo de usar, que nos da la información básica de los que

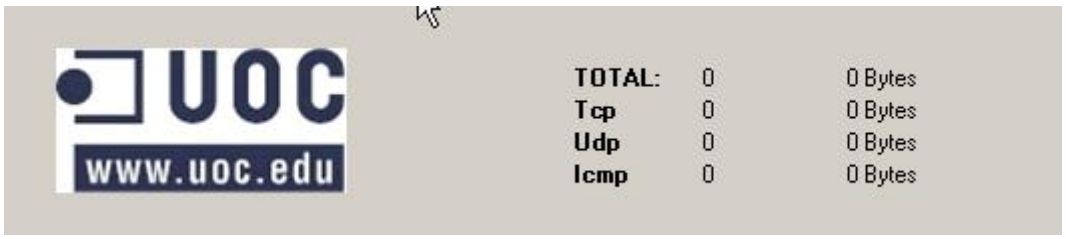
capturamos en nuestra red, y nos la muestra de forma clara. Como ya he comentado anteriormente, esto sería un primero paso para la creación de un sniffer mucho más complejo y parecido a los que existen en el mercado. Por lo que este es sencillo aunque conlleva toda la base para realizarle posibles mejoras y ampliaciones. Además al utilizar una librería independiente puede favorecer el realizar mejoras sin modificar el programa en si.

En la siguiente imagen podemos ver el único menú que tiene la aplicación "Opciones", en el que se muestra la opción donde hemos de indicarle de que Ips queremos capturar el tráfico. Y la siguiente imagen muestra la ventana para introducir Ips (de nuestra red) que podremos husmear.



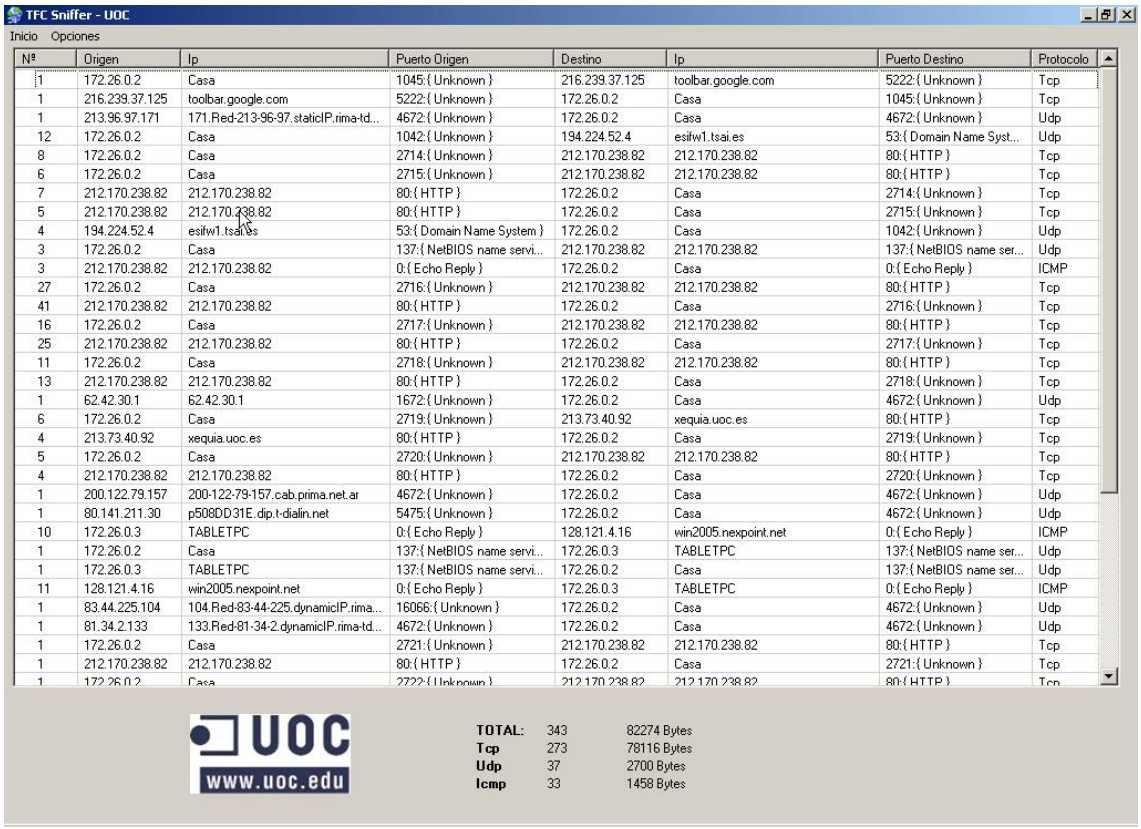


En la siguiente figura, podemos observar la parte baja de la pantalla, en la que se muestra el resumen de paquetes enviados y recibidos, y el total de bytes enviados y recibidos de esos paquetes,

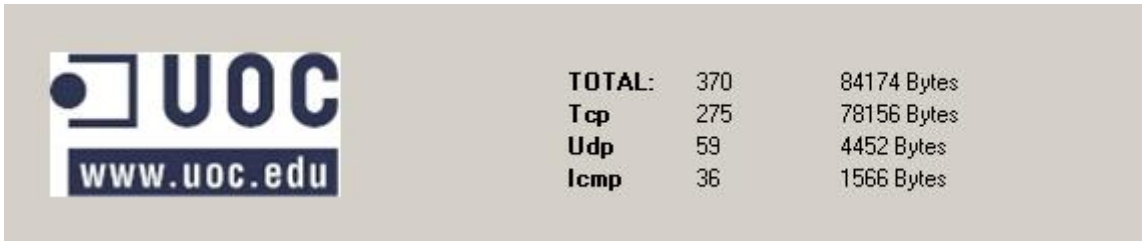


En principio solo distingo los paquetes TCP, UDP, ICMP.

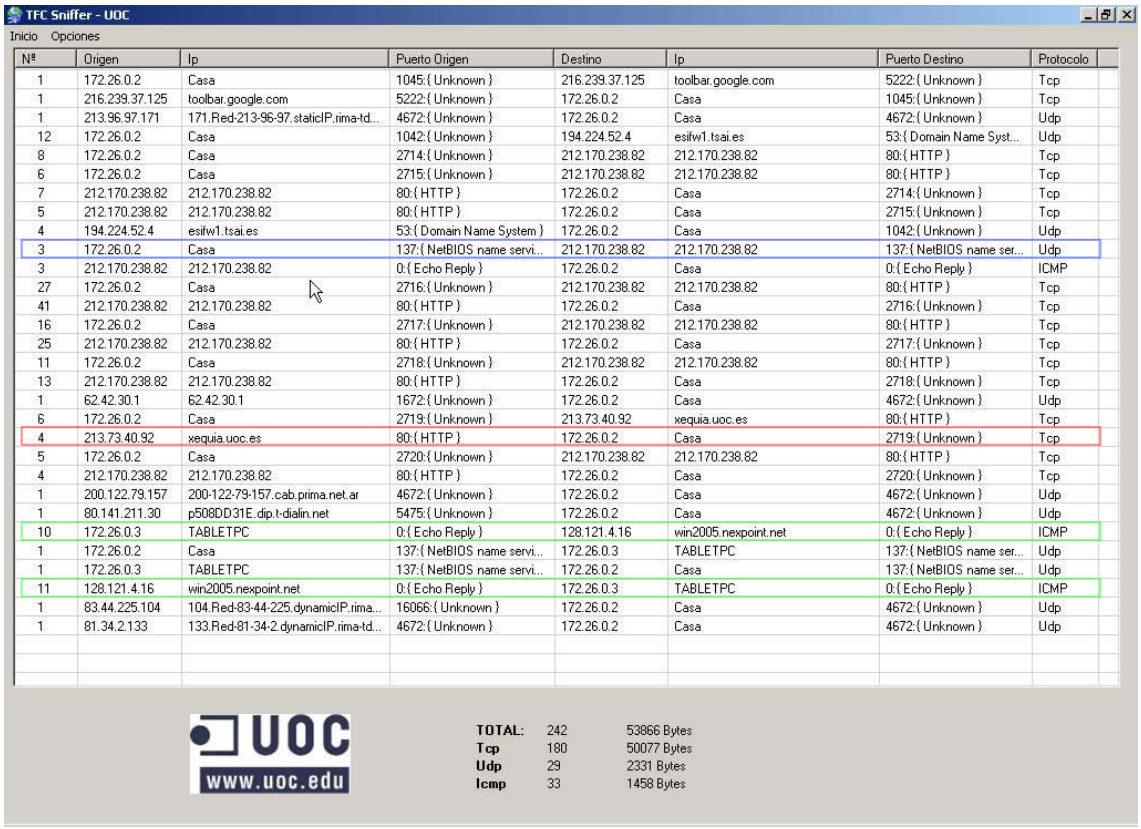
Y ahora la siguiente pantalla muestra la captura de información que realiza el programa, como se muestra y como queda reflejada. También muestro el resumen de abajo para que se vean los datos que ofrece la aplicación.



Nº	Origen	Ip	Puerto Origen	Destino	Ip	Puerto Destino	Protocolo
1	172.26.0.2	Casa	1045:{ Unknown }	216.239.37.125	toolbar.google.com	5222:{ Unknown }	Tcp
1	216.239.37.125	toolbar.google.com	5222:{ Unknown }	172.26.0.2	Casa	1045:{ Unknown }	Tcp
1	213.96.97.171	171.Red-213-96-97.staticIP.rima-td...	4672:{ Unknown }	172.26.0.2	Casa	4672:{ Unknown }	Udp
12	172.26.0.2	Casa	1042:{ Unknown }	194.224.52.4	esifw1.tsai.es	53:{ Domain Name Syst...	Udp
8	172.26.0.2	Casa	2714:{ Unknown }	212.170.238.82	212.170.238.82	80:{ HTTP }	Tcp
6	172.26.0.2	Casa	2715:{ Unknown }	212.170.238.82	212.170.238.82	80:{ HTTP }	Tcp
7	212.170.238.82	212.170.238.82	80:{ HTTP }	172.26.0.2	Casa	2714:{ Unknown }	Tcp
5	212.170.238.82	212.170.238.82	80:{ HTTP }	172.26.0.2	Casa	2715:{ Unknown }	Tcp
4	194.224.52.4	esifw1.tsai.es	53:{ Domain Name System }	172.26.0.2	Casa	1042:{ Unknown }	Udp
3	172.26.0.2	Casa	137:{ NetBIOS name servi...	212.170.238.82	212.170.238.82	137:{ NetBIOS name ser...	Udp
3	212.170.238.82	212.170.238.82	0:{ Echo Reply }	172.26.0.2	Casa	0:{ Echo Reply }	ICMP
27	172.26.0.2	Casa	2716:{ Unknown }	212.170.238.82	212.170.238.82	80:{ HTTP }	Tcp
41	212.170.238.82	212.170.238.82	80:{ HTTP }	172.26.0.2	Casa	2716:{ Unknown }	Tcp
16	172.26.0.2	Casa	2717:{ Unknown }	212.170.238.82	212.170.238.82	80:{ HTTP }	Tcp
25	212.170.238.82	212.170.238.82	80:{ HTTP }	172.26.0.2	Casa	2717:{ Unknown }	Tcp
11	172.26.0.2	Casa	2718:{ Unknown }	212.170.238.82	212.170.238.82	80:{ HTTP }	Tcp
13	212.170.238.82	212.170.238.82	80:{ HTTP }	172.26.0.2	Casa	2718:{ Unknown }	Tcp
1	62.42.30.1	62.42.30.1	1672:{ Unknown }	172.26.0.2	Casa	4672:{ Unknown }	Udp
6	172.26.0.2	Casa	2719:{ Unknown }	213.73.40.92	xequia.uoc.es	80:{ HTTP }	Tcp
4	213.73.40.92	xequia.uoc.es	80:{ HTTP }	172.26.0.2	Casa	2719:{ Unknown }	Tcp
5	172.26.0.2	Casa	2720:{ Unknown }	212.170.238.82	212.170.238.82	80:{ HTTP }	Tcp
4	212.170.238.82	212.170.238.82	80:{ HTTP }	172.26.0.2	Casa	2720:{ Unknown }	Tcp
1	200.122.79.157	200-122-79-157.cab.prima.net.ar	4672:{ Unknown }	172.26.0.2	Casa	4672:{ Unknown }	Udp
1	80.141.211.30	p508DD31E.dip.t-dialin.net	5475:{ Unknown }	172.26.0.2	Casa	4672:{ Unknown }	Udp
10	172.26.0.3	TABLETPC	0:{ Echo Reply }	128.121.4.16	win2005.nexpoint.net	0:{ Echo Reply }	ICMP
1	172.26.0.2	Casa	137:{ NetBIOS name servi...	172.26.0.3	TABLETPC	137:{ NetBIOS name ser...	Udp
1	172.26.0.3	TABLETPC	137:{ NetBIOS name servi...	172.26.0.2	Casa	137:{ NetBIOS name ser...	Udp
11	128.121.4.16	win2005.nexpoint.net	0:{ Echo Reply }	172.26.0.3	TABLETPC	0:{ Echo Reply }	ICMP
1	83.44.225.104	104.Red-83-44-225.dynamicIP.rima...	16066:{ Unknown }	172.26.0.2	Casa	4672:{ Unknown }	Udp
1	81.34.2.133	133.Red-81-34-2.dynamicIP.rima-td...	4672:{ Unknown }	172.26.0.2	Casa	4672:{ Unknown }	Udp
1	172.26.0.2	Casa	2721:{ Unknown }	212.170.238.82	212.170.238.82	80:{ HTTP }	Tcp
1	212.170.238.82	212.170.238.82	80:{ HTTP }	172.26.0.2	Casa	2721:{ Unknown }	Tcp
1	172.26.0.2	Casa	2722:{ Unknown }	212.170.238.82	212.170.238.82	80:{ HTTP }	Tcp



Y para finalizar muestro una captura en la que aparecen remarcados las capturas de dos ordenadores, (ip 172.26.0.2 y 172.26.0.3, se ha generado el trafico, con peticiones http, ping, y conexiones TCP y UDP con algún programa de intercambio para generar abundante tráfico), también aparecen marcados los diferentes paquetes que captura, (el ordenador 172.26.0.3 ha generado un PING por lo que él crea los paquetes ICMP).



Nº	Origen	Ip	Puerto Origen	Destino	Ip	Puerto Destino	Protocolo
1	172.26.0.2	Casa	1045:{ Unknown }	216.239.37.125	toolbar.google.com	5222:{ Unknown }	Tcp
1	216.239.37.125	toolbar.google.com	5222:{ Unknown }	172.26.0.2	Casa	1045:{ Unknown }	Tcp
1	213.96.97.171	171.Red-213-96-97.staticIP.rima-td...	4672:{ Unknown }	172.26.0.2	Casa	4672:{ Unknown }	Udp
12	172.26.0.2	Casa	1042:{ Unknown }	194.224.52.4	esifw1.tsai.es	53:{ Domain Name Syst...	Udp
8	172.26.0.2	Casa	2714:{ Unknown }	212.170.238.82	212.170.238.82	80:{ HTTP }	Tcp
6	172.26.0.2	Casa	2715:{ Unknown }	212.170.238.82	212.170.238.82	80:{ HTTP }	Tcp
7	212.170.238.82	212.170.238.82	80:{ HTTP }	172.26.0.2	Casa	2714:{ Unknown }	Tcp
5	212.170.238.82	212.170.238.82	80:{ HTTP }	172.26.0.2	Casa	2715:{ Unknown }	Tcp
4	194.224.52.4	esifw1.tsai.es	53:{ Domain Name System }	172.26.0.2	Casa	1042:{ Unknown }	Udp
3	172.26.0.2	Casa	137:{ NetBIOS name servi...	212.170.238.82	212.170.238.82	137:{ NetBIOS name ser...	Udp
3	212.170.238.82	212.170.238.82	0:{ Echo Reply }	172.26.0.2	Casa	0:{ Echo Reply }	ICMP
27	172.26.0.2	Casa	2716:{ Unknown }	212.170.238.82	212.170.238.82	80:{ HTTP }	Tcp
41	212.170.238.82	212.170.238.82	80:{ HTTP }	172.26.0.2	Casa	2716:{ Unknown }	Tcp
16	172.26.0.2	Casa	2717:{ Unknown }	212.170.238.82	212.170.238.82	80:{ HTTP }	Tcp
25	212.170.238.82	212.170.238.82	80:{ HTTP }	172.26.0.2	Casa	2717:{ Unknown }	Tcp
11	172.26.0.2	Casa	2718:{ Unknown }	212.170.238.82	212.170.238.82	80:{ HTTP }	Tcp
13	212.170.238.82	212.170.238.82	80:{ HTTP }	172.26.0.2	Casa	2718:{ Unknown }	Tcp
1	62.42.30.1	62.42.30.1	1672:{ Unknown }	172.26.0.2	Casa	4672:{ Unknown }	Udp
6	172.26.0.2	Casa	2719:{ Unknown }	213.73.40.92	xequia.uoc.es	80:{ HTTP }	Tcp
4	213.73.40.92	xequia.uoc.es	80:{ HTTP }	172.26.0.2	Casa	2719:{ Unknown }	Tcp
5	172.26.0.2	Casa	2720:{ Unknown }	212.170.238.82	212.170.238.82	80:{ HTTP }	Tcp
4	212.170.238.82	212.170.238.82	80:{ HTTP }	172.26.0.2	Casa	2720:{ Unknown }	Tcp
1	200.122.79.157	200-122-79-157.cab.prima.net.ar	4672:{ Unknown }	172.26.0.2	Casa	4672:{ Unknown }	Udp
1	80.141.211.30	p508DD31E.dip.t-dialin.net	5475:{ Unknown }	172.26.0.2	Casa	4672:{ Unknown }	Udp
10	172.26.0.3	TABLETPC	0:{ Echo Reply }	128.121.4.16	win2005.nexpoint.net	0:{ Echo Reply }	ICMP
1	172.26.0.2	Casa	137:{ NetBIOS name servi...	172.26.0.3	TABLETPC	137:{ NetBIOS name ser...	Udp
1	172.26.0.3	TABLETPC	137:{ NetBIOS name servi...	172.26.0.2	Casa	137:{ NetBIOS name ser...	Udp
11	128.121.4.16	win2005.nexpoint.net	0:{ Echo Reply }	172.26.0.3	TABLETPC	0:{ Echo Reply }	ICMP
1	83.44.225.104	104.Red-83-44-225.dynamicIP.rima...	16066:{ Unknown }	172.26.0.2	Casa	4672:{ Unknown }	Udp
1	81.34.2.133	133.Red-81-34-2.dynamicIP.rima-td...	4672:{ Unknown }	172.26.0.2	Casa	4672:{ Unknown }	Udp

El paquete marcado en azul corresponde a una conexión UDP al puerto 137 (localizado como puerto NetBIOS), también capturo los paquetes al puerto 53 TCP y UDP, y bastantes paquetes más, que he introducido para que reconozca el programa.

Los paquetes marcados en verde son los paquetes ICMP generados desde el ordenador 172.26.0.3 (un ping) y las contestaciones del otro sistema que van al ordenador 172.26.0.3.

Y el paquete marcado en rojo corresponde a un paquete TCP de conexión a la UOC, este en concreto es la respuesta de una conexión http del servidor de la UOC al ordenador 172.26.0.2.

Análisis de algún Sniffer actual

Antes de entrar en materia creo que es bueno introducir unos conceptos que ayuden a entender los siguientes puntos.

Hub / Switch:

Existen muchos tipos de redes, entre ellas, las que conectan los equipos mediante un hub y las que lo hacen mediante un switch. Supongamos que tenemos una mini red (LAN) con 3 equipos, como la de muchos particulares, pequeñas empresas, etc., conectados los tres a un hub, si el equipo 1 quiere abrir una carpeta (o recurso compartido) del equipo 2, la petición de entrar en esa carpeta sale del equipo 1, va al hub y éste la manda al equipo 2 y 3, pero como la petición era para el equipo 2, el 3 cuando la reciba simplemente la ignorará, y el 2 la recibirá y responderá, y con la respuesta volverá a suceder lo mismo.

En esta situación, una persona situada en el equipo 3 puede utilizar un sniffer, y dejarlo a la escucha, de esta forma puede ver toda la información que se intercambian los equipos 1 y 2, porque verá en el sniffer todos esos paquetes que no eran para el, sino para el equipo 2.

Si nuestra mini red tuviera un switch y no un hub el funcionamiento sería diferente, en la misma situación que antes, la petición del equipo 1 saldría hacia el switch, pero este la mandaría únicamente al equipo 2, puesto que va dirigida a él (esta es la diferencia con el hub) , y el 3 no se enteraría de nada. Ahora el sniffer del equipo 3 ya no sirve para nada.

Ip/Mac:

Cada equipo conectado a una red/internet, posee (al menos) una dirección IP y cada dispositivo de conexión posee una dirección MAC. La dirección IP es del tipo de X.X.X.X y sirve para distinguir un equipo de otro cuando estos están conectados (un equipo aislado, es decir, sin ningún tipo de conexión, también tiene IP, es la 127.0.0.1). La dirección MAC va "grabada" en cada dispositivo de conexión (tarjetas de red, routers, modems...) y en teoría no debe haber 2 iguales en el mundo, hasta hace poco se suponía que no se podían cambiar y desde no hace mucho existen técnicas para cambiarlas aunque no siempre funcionan.

Vamos a ver un ejemplo que nos ayudará a entender el párrafo siguiente, para el ejemplo de nuestra mini red, vamos a suponer que la IP del equipo 1 es 1.1.1.1 la del equipo 2 es 2.2.2.2 y la del 3 es 3.3.3.3, y que las direcciones MAC de sus tarjetas de red son MAC1, MAC2 y MAC3 respectivamente.

Arp Spoofing:

Es aquí donde entra el ARP (address resolution protocol). El ARP funciona mandando paquetes que hacen la siguiente consulta: "Si tu ip es X.X.X.X mándame tu dirección MAC", cuando un equipo recibe esta consulta, comprueba que la ip especificada es la suya y si es así responde enviando su dirección MAC, y el equipo que formuló la consulta, guardará la respuesta en la caché con la ip y la MAC (IP/MAC).

Lo importante es que estos paquetes son enviados a todos los equipos de la red, independientemente de si la red está conectada mediante un switch o un hub.

El arp-spoofing funciona mandando consultas y respuestas arp especialmente creadas.

De nuevo en nuestra mini red, si el equipo 3 manda respuestas arp al equipo 1 con la información IP/MAC manipulada, puede engañarlo y conseguir que si sale alguna información desde 1 hacia 2, ésta vaya en realidad hacia 3, y por lo tanto nos encontramos de nuevo el equipo 3 puede interceptar toda la comunicación entre 1 y 2 aunque la red disponga de un switch.

Hemos dicho un poco mas arriba "respuestas arp especialmente creadas", ¿qué

contendrían estas respuestas? simplemente, el equipo 3 construiría una respuesta en las que el contenido IP/MAC fuese 2.2.2.2/MAC3 (por supuesto en la realidad esto es mas complicado, pero creo que queda claro el funcionamiento), y la enviaría al equipo 1, por ejemplo, de esta manera esta respuesta queda guardada en la caché de 1, y creerá que a la ip 2.2.2.2 le corresponde la MAC3 en vez de la MAC2 (a esto se le llama envenenamiento ARP).

ETTERCAP

Cada día son más populares las técnicas de sniffing, arp-spoof, dns-spoof, man in the middle, etc., pero son muchos los que creen que están reservadas para los gurús de la red, y hasta hace poco era así, pero en esto como en todo, con el tiempo, internet ha hecho posible que se desarrollen rápidamente herramientas que hacen que lo que antes resultaba muy complicado, ahora sea cosa de niños, para descargar el programa basta con ir a la página http://ettercap.sourceforge.net/

He escogido el "ettercap" porque considero que es una de las mejores herramientas que se pueden encontrar para interceptar conexiones.

Este programa se ejecuta en Linux, aunque también puede instalarse en Windows con cygwin. Estos temas pueden resultar muy técnicos, aunque no voy a entrar en explicaciones profundas sobre el arp, sniff, topologías de red etc.... sólo trato de demostrar lo fácil que pude resultar un ataque teóricamente avanzado, es decir, cualquiera puede representar un peligro para nuestra seguridad.

¿Qué es?

Según sus autores:

"Ettercap es un sniffer/interceptor/logger para redes LAN con switchs, que soporta la disección activa y pasiva de muchos protocolos (incluso cifrados) e incluye muchas características para el análisis de la red y del host (anfitrión)".

Entre sus funciones, las más destacadas son las siguientes:

- Injection de caracteres en una conexión establecida emulando comandos o respuestas mientras la conexión está activa.
- Compatibilidad con SSH1: Puede interceptar users y passwords incluso en conexiones "seguras" con SSH.

- Compatibilidad con HTTPS: Intercepta conexiones mediante http SSL (supuestamente seguras) incluso si se establecen a través de un proxy. *Intercepta tráfico remoto mediante un túnel GRE: Si la conexión se establece mediante un túnel GRE con un router Cisco, puede interceptarla y crear un ataque "Man in the Middle".
- "Man in the Middle" contra túneles PPTP (Point-to-Point Tunneling Protocol).
- Soporte para Plug-ins.

Listado de plugins:

- Colector de contraseñas en : TELNET, FTP, POP, RLOGIN, SSH1, ICQ, SMB, MySQL, HTTP, NNTP, X11, NAPSTER, IRC, RIP, BGP, SOCKS 5, IMAP 4, VNC, LDAP, NFS, SNMP, HALF LIFE, QUAKE 3, MSN, YMSG.
- Filtrado y sustitución de paquetes.
- OS fingerprint: es decir, detección del sistema operativo remoto.
- Mata conexiones.
- Escáner de LAN: hosts, puertos abiertos, servicios...
- Busca otros envenenamientos en la misma red.
- Port Stealing (robo de puertos): es un nuevo método para el sniff en redes con switch, sin envenenamiento ARP".

Esta información está sacada de la página oficial del Ettercap y nos da una idea de lo potente que es esta herramienta.

Nuestra herramienta para empezar no va a ser tan potente ni nada parecido pero con el diseño que crearé se podrá llegar a generar una aplicación igual o más potente que esta.

Ahora y para entender mejor el sniffer paso a explicar un poco su uso, puesto que he obtenido mucha información de este sniffer.

ETHEREAL

También y para no faltar, expongo un vistazo de este otro sniffer de red, creo que es de los más usados porque es mucho más sencillo e intuitivo y nosotros mismos en la carrera es el que usamos.

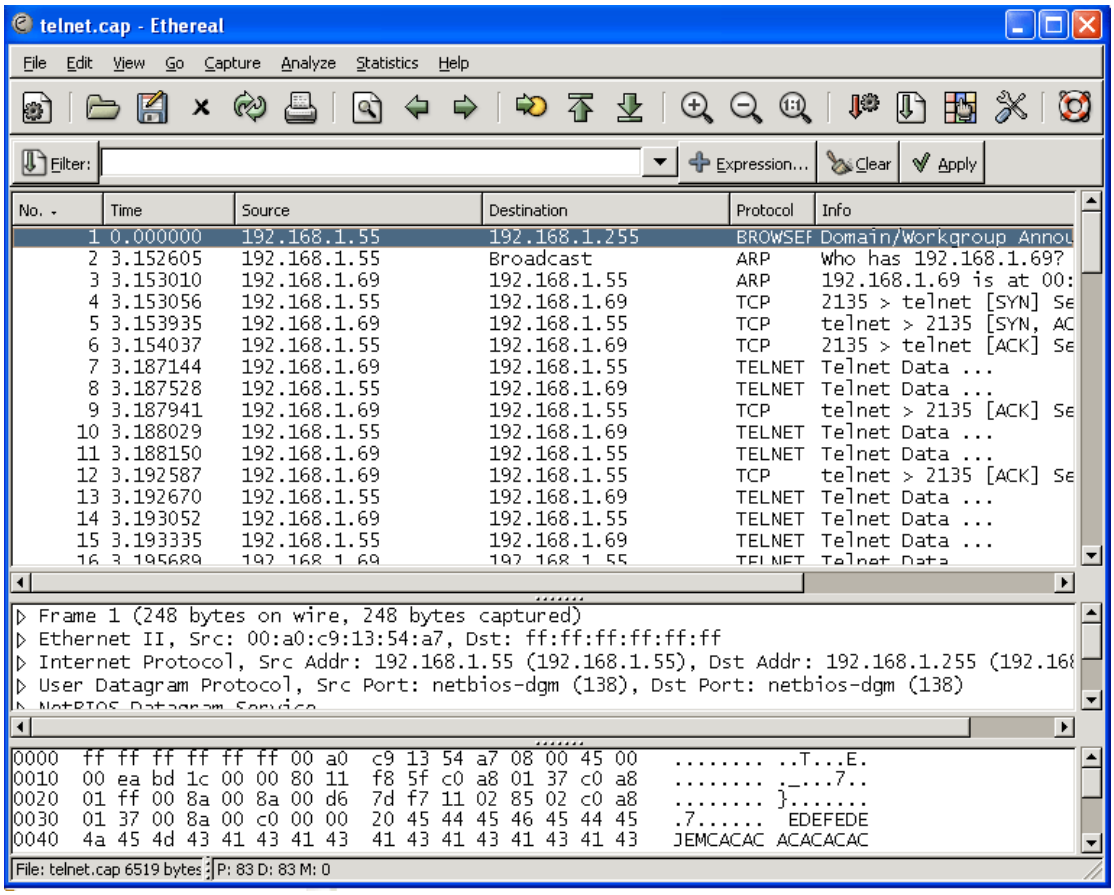
Pero ¿Qué es Ethereal? es un analizador de protocolos, utilizado para realizar análisis y solucionar problemas en redes de comunicaciones, para desarrollo de software y protocolos, y como una herramienta didáctica para educación. Cuenta con todas las características estándar de un analizador de protocolos.

La funcionalidad que provee es similar a la de tcpdump, pero añade una interfaz gráfica y muchas opciones de organización y filtrado de información. Así, permite ver todo el tráfico que pasa a través de una red (usualmente una red Ethernet, aunque soporta algunas otras) estableciendo la configuración en modo promiscuo. Ethereal está desarrollado bajo la licencia open source, y se ejecuta sobre la mayoría de sistemas operativos Unix y compatibles, incluyendo Linux, Solaris, FreeBSD, NetBSD, OpenBSD, y Mac OS X, así como en Microsoft Windows.

Los aspectos más importantes de Ethereal son:

- Es mantenido bajo la Licencia GPL
- Trabaja tanto en modo promiscuo como en modo no promiscuo
- Puede capturar datos de la red o leer datos almacenados en un archivo (de una captura previa)
- Tiene una interfaz muy flexible
- Capacidades de filtrado muy ricas
- Soporta el formato estándar de archivos tcpdump
- Reconstrucción de sesiones TCP
- Se ejecuta en más de 20 plataformas
- Soporta más de 480 protocolos
- Puede leer archivos de captura de más de 20 productos

Con este mini análisis creo que ya puedo tener claro que es lo que quiero hacer y lo que se podría hacer con un sniffer.


telnet.cap - Ethereal
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply
No. Time Source Destination Protocol Info
1 0.000000 192.168.1.55 192.168.1.255 BROWSEF Domain/Workgroup Annou
2 3.152605 192.168.1.55 Broadcast ARP Who has 192.168.1.69?
3 3.153010 192.168.1.69 192.168.1.55 ARP 192.168.1.69 is at 00:
4 3.153056 192.168.1.55 192.168.1.69 TCP 2135 > telnet [SYN] Se
5 3.153935 192.168.1.69 192.168.1.55 TCP telnet > 2135 [SYN, AC
6 3.154037 192.168.1.55 192.168.1.69 TCP 2135 > telnet [ACK] Se
7 3.187144 192.168.1.69 192.168.1.55 TELNET Telnet Data ...
8 3.187528 192.168.1.55 192.168.1.69 TELNET Telnet Data ...
9 3.187941 192.168.1.69 192.168.1.55 TCP telnet > 2135 [ACK] Se
10 3.188029 192.168.1.55 192.168.1.69 TELNET Telnet Data ...
11 3.188150 192.168.1.69 192.168.1.55 TELNET Telnet Data ...
12 3.192587 192.168.1.69 192.168.1.55 TCP telnet > 2135 [ACK] Se
13 3.192670 192.168.1.55 192.168.1.69 TELNET Telnet Data ...
14 3.193052 192.168.1.69 192.168.1.55 TELNET Telnet Data ...
15 3.193335 192.168.1.55 192.168.1.69 TELNET Telnet Data ...
Frame 1 (248 bytes on wire, 248 bytes captured)
Ethernet II, Src: 00:a0:c9:13:54:a7, Dst: ff:ff:ff:ff:ff:ff
Internet Protocol, Src Addr: 192.168.1.55 (192.168.1.55), Dst Addr: 192.168.1.255 (192.168
User Datagram Protocol, Src Port: netbios-dgm (138), Dst Port: netbios-dgm (138)
0000 ff ff ff ff ff ff 00 a0 c9 13 54 a7 08 00 45 00 ....T...E.
0010 00 ea bd 1c 00 00 80 11 f8 5f c0 a8 01 37 c0 a8 ..... ._...7..
0020 01 ff 00 8a 00 8a 00 d6 7d f7 11 02 85 02 c0 a8 ..... }.....
0030 01 37 00 8a 00 c0 00 00 20 45 44 45 46 45 44 45 .7..... EDEFEDE
0040 4a 45 4d 43 41 43 41 43 41 43 41 43 41 43 41 43 JEMCACAC ACACACAC
File: telnet.cap 6519 bytes
P: 83 D: 83 M: 0

Implementación del programa

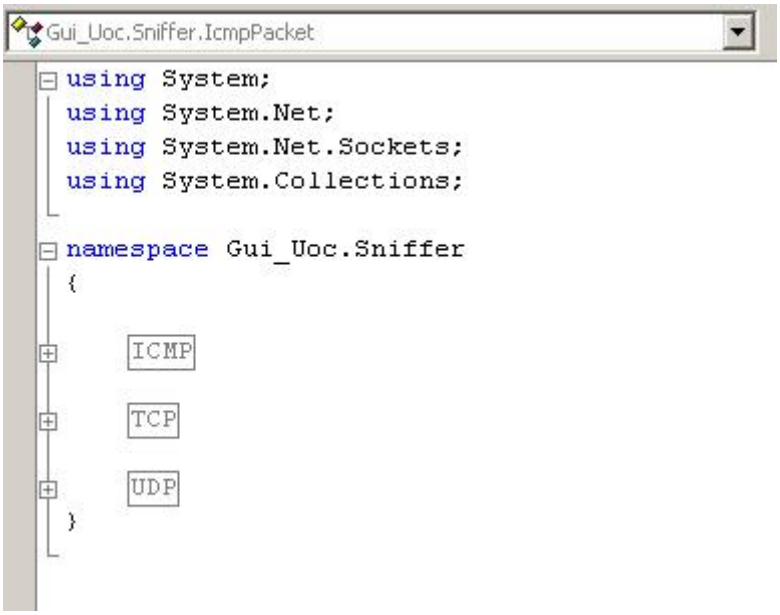
Definición y Desarrollo

La técnica de programación usada es la de orientación a objetos. El lenguaje es C#.NET v1.1 y el entorno es Visual Studio 2003 Profesional.

Ahora voy a exponer las técnicas generales que he usado para el desarrollo de la aplicación.
Para guardar los datos necesarios en memoria he utilizado de forma intensiva los Arrays generales para pocos datos, pero para mantener guardada y accesible la información de los paquetes, me decidí a utilizar HashTables, un tipo de colección que nos permite guardar pares de información, a la que podemos acceder a ella mediante la clave, además de eliminar registros, disponer de funciones de comparación, vaciado completo e inicialización. En la imagen tenemos algunas de las inicializaciones que hacemos, a las variables que necesitamos, como hashtable.

```
public Hashtable Contador_ =null;
private Hashtable Contadores_=null;
private Hashtable SnifActual_=null;
public Hashtable DataTable_ =null;
```

El diseño se basa en la creación de clases con los “tipos” que usaré (una clase para cada paquete con sus características, TCP, UDP, ICMP), por comodidad, he creado las 3 clases dentro de otra más (que las contiene) para poder agrupar y acceder más rápidamente.



Y lo he agrupado en regiones (que es la forma de poder "ocultar" partes del código con el símbolo + que aparece, para mostrar solamente la necesaria).

Ahora mostraré las clases ICMP, UDP y TCP (solo mostraré el inicio por facilidad, puesto que son clases bastante largas), estas clases están preparadas con casi todos los campos que trae el datagrama de este tipo, los he obtenido de los RFCs oficiales además de variada documentación encontrada en internet.

```
    #region UDP

    /// <summary>
    /// Clase que encapsula todas las propiedades de los paquetes UDP y el
    /// acceso a ellas
    /// </summary>
    public class UdpDatagram
    {
        private IPEndPoint Source_ = null;
        private IPEndPoint Destination_ = null;
        private byte[]     Data_ = null;

        /// <summary>
        /// Establece/Devuelve el puerto de origen
        /// </summary>
        public int SourcePort
        {
            get { return Source_.Port; }
        }

        /// <summary>
        /// Establece/Devuelve el puerto de destino
        /// </summary>
        public int DestinationPort
        {
            get { return Destination_.Port; }
        }

        /// <summary>
        /// Establece/Devuelve la IP de origen
```

Podemos ver las variables inicializadas, y las primeras funciones GET/SET.
Ahora mostraré la misma clase de TCP e ICMP. La clase TCP es la que más variables tiene, puesto que su datagrama es el más complejo, aunque aparezcan todas las secciones del datagrama, no están implementadas, está listo para poder programarse, y será una de las mejoras, (el detalle de cada paquete), para una futura versión.

```
#region TCP

/// <summary>
/// Clase que encapsula todas las propiedades de los paquetes TCP
/// y el acceso a ellas
/// </summary>
public class TcpPacket
{
    private IPEndPoint Source_ = null;
    private IPEndPoint Destination_ = null;
    private uint Sequence_ = 0;
    private uint Acknowledgement_ = 0;
    private int  DataOffset_ = 0;
    private bool Urgent_ = false;
    private bool Push_ = false;
    private bool Ack_ = false;
    private bool Fin_ = false;
    private bool Syn_ = false;
    private bool Reset_ = false;
    private int WindowSize_ = 0;
    private int Checksum_ = 0;
    private int UrgentPointer_ = 0;
    private byte[] Data_ = null;
    private byte[] Options_ = null;
    private int  Length_ = 0;

    /// <summary>
    /// Constructor de la clase
    /// </summary>
    public TcpPacket()
    {
```

Y el paquete ICMP, que aunque también tiene su complejidad y múltiples opciones me he intentado limitar a las funciones básicas.

```
#region ICMP

/// <summary>
/// Clase que define los paquetes Icmp.
/// </summary>
public class IcmpPacket
{
    private IPEndPoint Source_ = null;
    private IPEndPoint Destination_ = null;
    private byte Type_;
    private byte Code_;
    private UInt16 Checksum_;
    private int DataSize_;
    private byte[] Data_ = new byte[1024];

    /// <summary>
    /// Constructor de la clase
    /// </summary>
    public IcmpPacket()
    {

    }

    /// <summary>
    /// Establece/Devuelve el campo Origen
    /// </summary>
    public IPEndPoint Source
    {
        get { return Source_; }
        set { Source_ = value; }
    }
```

Además hago uso de threads (2 en este caso) para poder ir capturando los paquetes y actualizando los contadores de información y mientras que el programa sea manejable. Dichos threads se ejecutan desde la clase principal, que es la que los crea. Dejando su control a clases que más adelante explicaré.

```
// Inicio del Thread que gestiona los contadores, los actualiza.
thContadores = new Thread(new ThreadStart(counterBytes));
thContadores.IsBackground = true;
thContadores.Start();

// Inicio del Thread que carga los paquetes en el ListView
thSniffer = new Thread(new ThreadStart(loadPackets));
thSniffer.IsBackground = true;
thSniffer.Start();
```

Partiendo de estos primeros esbozos he generado el programa totalmente funcional. Pero lo más complicado ha sido la obtención de los paquetes.

Como anteriormente expuse un sniffer se basa en la captura de todos los paquetes que lleguen (le pertenezcan o no) a la interfaz de red. ¿Cómo se consigue eso? Ya comenté a modo de teoría lo que son los RawSockets y para que sirven. Ahora voy a explicar las funciones que hacen que eso sea posible en mi programa.

Además de crear las clases necesarias hay que juntarlo con el GUI para darles uso. Nada más cargar la clase principal creo las variables que necesito, y obtengo la IP de la máquina donde está corriendo (una vez empiece se pueden agregar más Ips),

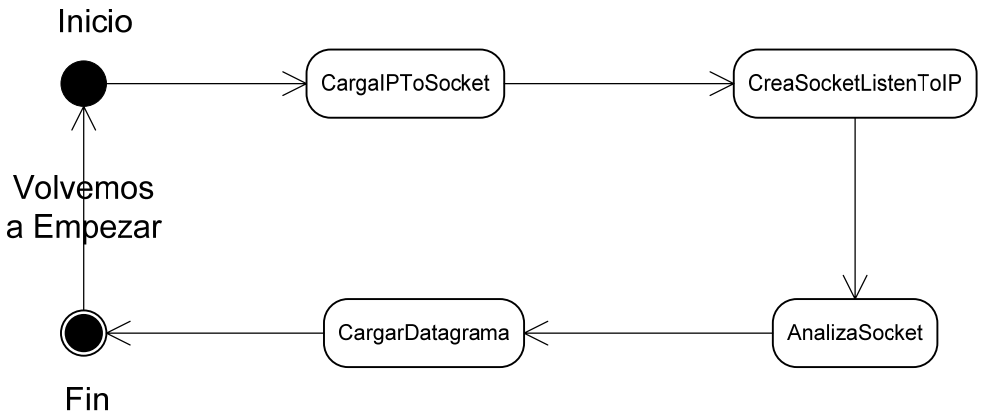
```
/// Clase principal del programa
/// </summary>
public Gui_Uoc()
{
    InitializeComponent();
    DataTable_= new Hashtable();
    Contador_=new Hashtable();
    SnifActual_= new Hashtable();
    Contadores_=new Hashtable();
    lstwMain.FullRowSelect=true;
    lstwMain.GridLines=true;
    tempDatagram = new Sniffer.cDatagrama();
    thisIp =Dns.GetHostByName(Dns.GetHostName()).AddressList[0].ToString();
    bool validable = true;

    while (validable)
    {
        if ( thisIp != null )
        {
            Socket_ = new Sniffer.HusmearSocket();
            try
            {
                Socket_.Sniff(thisIp);
                Socket_.IPs.Add(thisIp);
                validable=false;
            }
            catch(Exception e)
            {
                HandleSnifferError(e);
            }
            LoadOptions();
```

La variable thisIP obtiene la IP de la máquina donde corre el programa (utilizo funciones básicas que trae el propio framework. Luego en el bucle WHILE, ahí es donde comienza todo el programa a funcionar, lo primero que hago es comprobar que he obtenido una IP, y posteriormente creo un objeto “HumearSocket ()”, que como ya dije es el que tiene todo el control de los sockets, y es donde tengo las

funciones, una vez creado, añado la IP al Sniff, y a la lista de Ips. Valido a false la variable para salirme del bucle y cargo las opciones por defecto y debajo de este código también es donde cargo los dos threads para que vayan actualizando la información en la lista y los contadores.

Hay que tener en cuenta, que el programa lo que hace es, obtiene un socket, lo analiza, y lo añade a una HashTable, y suma a los contadores lo que tenga que sumar, posteriormente los threads ejecutándose en background van mostrando dicha información. Haciendo de esta forma el programa manejable aunque sigan ejecutándose acciones hasta que se apague.



Ahora vemos la función Sniff, esta función a la que le hemos de pasar la IP (como hemos visto en la función que carga Gui_Uoc, le pasa directamente la IP además de añadirla a la lista de Ips a husmear; tras pasarle la IP creamos el socket con las propiedades que necesitamos, es importante destacar el tipo de Sockets a capturar (RAW), creamos un buffer más que sobrado, y creamos una variable SocketGest que dispone de las funciones de control del Buffer y de almacenamiento del Socket; y comenzamos el proceso, primeramente asociamos un objeto socket a una IP, la clase SetupSocket, a la que le pasamos el “Socket” nos lo carga con los primeros datos que extrae del buffer, comprobamos en la lista de sockets abiertos (SocketMap_) si existe ya nuestra Ip, sino existe la añadimos, y comenzamos a recibir.

```
public void Sniff(String ip)
{
    IP=ip;
    Socket socket = new Socket(AddressFamily.InterNetwork,SocketType.Raw,ProtocolType.IP)
    byte[] buffer = new byte[2048];
    SocketGest socketgest = new SocketGest(socket,buffer);
    socket.Blocking = true;
    socket.Bind(new IPEndPoint(IPAddress.Parse(ip),0));
    this.SetupSocket(socket);
    if ( SocketMap_.Contains(ip) )
    {
        MessageBox.Show("Ya está abierto un socket para esa IP ","Error",MessageBoxButtor
    }
    else { SocketMap_.Add(ip,socketgest);}
    try
    {
        socket.BeginReceive(buffer,0,buffer.Length,SocketFlags.None,new AsyncCallback(thi
    }
    catch ( Exception e )
    {
        MessageBox.Show("No puede comenzar a Recibir " + e.ToString(),"Error",MessageBoxE
    }
}
```

Gracias a esta clase ya comenzamos a recibir los sockets, que con la función AsyncCallback, posteriormente analizaremos y asignaremos sus propiedades, los guardaremos y los Threads se encargarán de listarlos.

Detectando Sniffers en nuestra red

Redes conmutadas y no conmutadas.

Vamos a tratar aquí, principalmente, la detección de sniffers en nuestra red desde el escenario más básico posible. Este escenario sería una subred o red no conmutada. Aunque más adelante nos introduciremos brevemente en la escucha en redes conmutadas o basadas en switches y herramientas de detección en este tipo de redes.

Los *sniffers no son fáciles de detectar ni de combatir*, ya que se trata de programas que trabajan en modo pasivo. Las técnicas que se tratan aquí, por tanto, no son totalmente fiables, aunque en algunos casos si suponen una gran aproximación al descubrimiento de este tipo de software.

Detección en sistemas UNIX/LINUX

Técnicas de Detección

- **El test DNS**

En este método, la herramienta de detección en sí misma está en modo promiscuo. Creamos numerosas conexiones *TCP falsas* en nuestro segmento de red, esperando un sniffer pobremente escrito para atrapar estas conexiones y resolver la dirección IP de los inexistentes hosts.

Algunos sniffers realizan búsquedas inversas DNS en los paquetes que capturan. Cuando se realiza una *búsqueda inversa DNS*, una utilidad de detección de sniffers *"huele"* la petición de las operaciones de búsqueda para ver si el objetivo es aquel que realiza la petición del *host inexistente*.

- **El Test del Ping**

Este método confía en un problema en el núcleo de la máquina receptora. Podemos construir una petición tipo *"ICMP echo"* con la *dirección IP de la máquina*

sospechosa de hospedar un sniffer, pero con una *dirección MAC deliberadamente errónea*.

Enviamos un paquete *"ICMP echo"* al objetivo con la dirección IP correcta, pero con una *dirección de hardware de destino distinta.*

La mayoría de los sistemas desatenderán este paquete ya que su dirección MAC es incorrecta. Pero en algunos sistemas Linux, NetBSD y NT, puesto que el NIC está en modo promiscuo, el sniffer asirá este paquete de la red como paquete legítimo y responderá por consiguiente. Si el blanco en cuestión responde a nuestra petición, sabremos que está en modo promiscuo.

Un atacante avanzado puede poner al día sus sniffers para filtrar tales paquetes para que parezca que el NIC no hubiera estado en modo promiscuo.

- **El Test ICMP. *Ping de Latencia.***

En éste método, hacemos *ping al blanco* y anotamos el Round Trip Time *(RTT, retardo de ida y vuelta o tiempo de latencia).*

Creamos centenares de falsas conexiones TCP en nuestro segmento de red en un período de tiempo muy corto. Esperamos que el sniffer esté procesando estos paquetes a razón de que el tiempo de latencia incremente.

Entonces hacemos *ping otra vez*, y *comparamos el RTT* esta vez con el de la primera vez.

Después de una serie de tests y medias, podemos concluir o no si un sniffer está realmente funcionando en el objetivo o no.

- **El test ARP**

Podemos enviar una *petición ARP* a nuestro objetivo con toda la información rápida excepto con una *dirección hardware de destino errónea*.

Una máquina que no esté en modo promiscuo nunca verá este paquete, puesto que no era destinado a ellos, por lo tanto no contestará.

Si una máquina está en modo promiscuo, la petición ARP sería considerada y el núcleo la procesaría y *contestaría*. Por la máquina que contesta, sabemos que estamos en modo promiscuo.

- **El test Etherping**

Enviamos un "ping echo" al host a testear con una IP de destino correcta y dirección MAC falseada.

Si el host responde, es que su interfaz está en modo promiscuo, es decir, existe un sniffer a la escucha y activo.

Otras formas de Detectar Sniffers

- Detectar y controlar los logs que suelen generar los sniffers.
- Detectar y controlar las conexiones al exterior.
- Monitorizar los programas que acceden al dispositivo de red.
- Normalmente una interface en modo promiscuo, queda reflejada en el fichero de logs:

Detección en Sistemas Windows

- **PROMISCAN**

"PromiScan *(www.securityfriday.com)* es una utilidad de distribución gratuita diseñada para dar caza a los nodos promiscuos en una LAN rápidamente y *sin crear una carga pesada en la red*.

Hay que tener en cuenta que localizar un nodo promiscuo es una tarea ardua, y que en muchos casos el resultado es incierto; no obstante, PromiScan consigue mostrar cada uno de esos nodos de una manera transparente, claramente visible.

Para usar PromiScan es necesario contar con Windows 2000 Professional y haber instalado previamente el controlador WinPcap."

http://www.securityfriday.com/ToolD...iscan_003.html

- **PROMISDETECT**

http://www.ntsecurity.nu/cgi-bin/do...scdetect.exe.pl

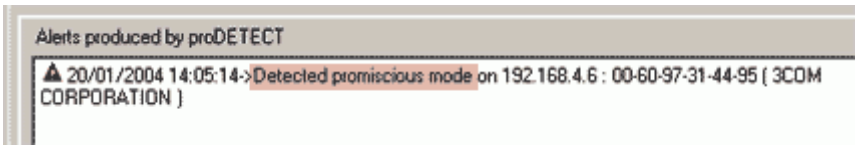
http://ntsecurity.nu/downloads/promiscdetect.exe

- **ProDETECT 0.2 BETA**

http://sourceforge.net/projects/prodetect/

http://prdownloads.sourceforge.net/prodetect/proi386.exe?download

Alerta de ProDETECT:



Detección en Redes Conmutadas

En redes commutadas o que hagan uso de switches, la técnica de ARP poisoning o *envenenamiento arp* es la más efectiva. Esta técnica consiste, muy brevemente, en *modificar (envenenar) la tabla ARP de los host involucrados* en el ataque para que éstos envíen a la red tramas Ethernet con destino la MAC del atacante. Esto significa que el switch entregará los datos de las comunicaciones a dicho host. Para evitar el refresco de la caché ARP es necesario el envío constante de *arp-reply*.

Una posible solución o defensa sería el uso de MACs estáticas, con el fin de que no puedan ser modificadas, aunque en algunos sistemas Windows esto no es eficiente al 100 por 100.

- **ARPWatch**

En sistemas Linux la herramienta ARPWatch *(http://www-nrg.ee.lbl.gov/)* nos puede servir para detectar el uso del envenenamiento ARP en nuestro sistema. Con ARPWatch podemos comprobar la correspondencia entre pares IP-MAC *(Ethernet)*.

En el caso de que un cambio en un par se produzca *(esto es, se escuche en el interfaz de red del sistema)*, ARPWatch envía un correo de notificación del suceso a la cuenta root o administrador del sistema con un mensaje tipo *"FLIP FLOP o Change ethernet address"*. También podemos monitorizar la existencia de nuevos host *(aparición de una nueva MAC en la red).*

- **WinARP Watch v1.0**

Una herramienta similar a ARPwatch pero para sistemas Windows la encontramos en WinARP Watch v1.0 *(http://www.securityfocus.com/data/tools/warpwatch.zip)*. Esta herramienta no enviará correo alguno a ningún administrador, pero nos tendrá puntualmente informados sobre la *caché ARP*, las *correspondencias IP/MAC*, cualquier nuevo par que se añada a la caché, etc.

Y cada día aparecen nuevos programas que pueden sernos útiles a la hora de detectar los sniffers en nuestra red.

Bibliografía

- Definición de Sniffers General
http://www.fortunecity.es/imaginapoder/artes/368/escuela/telecom/sniffer.htm
- Todo sobre el sniffing http://www.badopi.org/node/23
- CSharpSniffer (Husmeador de tráfico en adaptador de red local):
http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/MTJ_2068.asp
- Introducción a los sniffer's y anti-sniffers:
http://etigol.todo-linux.com/docs_sniffers.php
- TCP/IP Raw Sockets:
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winsock/winsock/tcp_ip_raw_sockets_2.asp
- Que es un Socket: http://www.arrakis.es/~dmrq/beej/theory.html
- Información de winsock:
http://www.frostbytes.com/~jimf/papers/sockets/winsock.html
- PROTOCOLOS:
http://www.saulo.net/pub/tcpip/
http://eia.udg.es/~atm/tcp-ip/
http://neo.lcc.uma.es/evirtual/cdd/tutorial/red/arp.html
- Dsniff: http://www.monkey.org/~dugsong/dsniff/
- Sniffit: http://reptile.rug.ac.be/~coder/sniffit/sniffit.html
- Net-tools: http://www.tazenda.demon.co.uk/phil/net-tools/
- 7a69ezine: http://www.7a69ezine.org/
- Antisniff: http://www.l0pht.com/antisniff/
http://www.securityfocus.com/data/tools/anti_sniff_researchv1-1-2.tar.gz
- Sentinel: http://www.packetfactory.net/Projects/sentinel/
- TCPDUMP: http://www.tcpdump.org
- Sniffing-faq: http://www.robertgraham.com/pubs/sniffing-faq.html
- Sniffing (network wiretap, sniffer) FAQ:
http://cs.baylor.edu/~donahoo/tools/sniffer/sniffingFAQ.htm

Anexos

Como compilar el programa sin el entorno de programación Visual Studio

Para poder compilar el programa es necesario tener instalado el SDK framework .NET 1.0 y el 1.1, pero la versión SDK y la versión redistribuible, y se puede descargar desde aquí:

http://msdn.microsoft.com/netframework/downloads/updates/default.aspx

Aquí se debe seleccionar el idioma del sistema operativo que tengamos.

Una vez hecho esto y para realizar una compilación rápida descargamos un IDE de programación gratuito:

http://www.icsharpcode.net/OpenSource/SD/Download/

Descargado e instalado dicho entorno (hay que descargar la versión primera la que es preferente para 1.1. Debemos de importar el proyecto.

¿Pero donde está el proyecto con el código fuente? Pues está en una carpeta en donde se haya instalado el programa. Ahora ejecutamos el IcsharpCode y en el menú Archivo le damos a Importar Proyecto. En Importar proyecto seleccionamos el archivo TFC_UOC.csproj que está dentro de la carpeta con el código. Le damos a aceptar y el proyecto estaría preparado en el nuevo entorno.
Ahora solo haría falta recompilarlo si se desea y tendríamos el código fuente completo y disponible para realizar cualquier cambio.