Mango Computer c.a

Manual de Usuario

Mango Sniffer 2 (MS2)

Creador por: José Andrés Morales Linares 2009

comprasmangocomputer@gmail.com

Índice

Contenido

Pag.

Hack, Hacker y Cracker

Los términos hacker y hack tienen connotaciones positivas e, irónicamente, también negativas. Los programadores informáticos suelen usar las hacking y hacker para expresar admiración por el trabajo de un desarrollador de software calificado, pero también se puede utilizar en un sentido negativo para describir una solución rápida pero poco elegante a un problema. Algunos desaprueban el uso del hacking como un sinónimo de cracker, enmarcado contraste con el resto del mundo, en el que la palabra hacker se utiliza normalmente para describir a alguien que "hackea" un sistema con el fin de eludir o desactivar las medidas de seguridad.

Controversia y Ambigüedad

En un principio se utilizaba "hack" como verbo para expresar "perder el tiempo" (e.j. "Puedo hackear con el ordenador"), el significado del término ha cambiado a lo largo de décadas desde que empezó a utilizarse en un contexto informático. Como su uso se ha extendido más ampliamente, el significado primario de la palabra, por parte de los nuevos usuarios, ha pasado a uno que entra en conflicto con el énfasis original.

Activismo

Desde el año 2002-2003, se ha ido configurando una perspectiva más amplia del hacker, pero con una orientación a su integración al hacktivismo en tanto movimiento. Aparecen espacios autónomos denominados hacklab y los hackmeeting como instancias de diálogo de hackers. Desde esta perspectiva, se entiende al hacker como una persona que es parte de una conciencia colectiva que promueve la libertad del conocimiento y la justicia social.

En este caso, los roles de un hacker pueden entenderse en cuatro aspectos:

- Apoyar procesos de apropiación social o comunitaria de las tecnologías.
- Poner a disposición del dominio público el manejo técnico y destrezas alcanzadas personal o grupalmente.
- Crear nuevos sistemas, herramientas y aplicaciones técnicas y tecnológicas para ponerlas a disposición del dominio público.
- Realizar acciones de hacktivismo tecnológico con el fin de liberar espacios y defender el conocimiento común, o mancomunal.

> “El mundo real no solo es blanco y negro, sino infinidad de escalas de grises entre el bien y el mal, entre blanco y el negro, entre el inicio y final, entre el cosmos y el caos, entre el ***hacker y el cracker”.***
>
> ***José Andrés Morales***

¿Qué es un Sniffer de Red?

Un sniffer es un programa de captura de las tramas de red.

Es algo común que, el medio de transmisión (cable coaxial, UTP, fibra óptica etc.) sea compartido por varias computadoras y dispositivos de red, lo que hace posible que un ordenador capture las tramas de información no destinadas a él. Para conseguir esto el sniffer le dice a la computadora que deje de ignorar todo el tráfico no destinado al equipo y le ponga atención, esto es conocido como poner en estado "*promiscuo*" a la NIC (Network Interface Card).

En la actualidad la seguridad en las redes es de vital importancia, ya que toda la información que se transmite a través de éstas muchas veces puede ser utilizada para fines de lucro o realizar delitos electrónicos.

Una vez que la NIC está en este estado se necesitarán los privilegios administrativos o de root, de ésta manera la computadora será capaz de ver todos los datos transmitidos. Es entonces cuando el programa comienza a hacer una lectura de toda la información entrante al PC por la tarjeta de red. Con esto el sniffer conseguirá observar el equipo de origen, el equipo de destino, número de puerto, entre otros, en resumen puede ver la información intercambiada entre dos computadoras.

El uso que se les den a éste tipo de aplicaciones es algo importante de señalar, ya que gracias a ellos podemos ayudar a que nuestra Red tenga más seguridad, hacer pruebas y así poder tener un muy buen resultado, el problema viene cuando otros usuarios lo utilizan con fines de delitos electrónicos, ya que con éste tipo de herramientas se puede obtener información confidencial.

Mango Sniffer 2 - MS2

Mango Sniffer 2, o como cariñosamente lo ha bautizado su creador "MS2". Es un programa de computadora escrito en Lenguaje C puro, que utiliza una biblioteca de apoyo llamada "libpcap". MS2 es un programa denominado Sniffer (Palabra en ingles que literalmente debería traducirle como "husmeador"). Mas concretamente MS2 es un herramienta en línea de comandos cuya utilidad principal es analizar el tráfico que circula por la red. Permite al usuario capturar y mostrar a tiempo real los paquetes transmitidos y recibidos en la red a la cual el ordenador está conectado.

Los principales usos que se le pueden dar son:

- Para la enseñanza, en cursos avanzados de topologías de red y redes en general.
- Captura de contraseñas enviadas sin cifrar y nombres de usuario de la red. Esta capacidad es utilizada en muchas ocasiones por atacantes (cracker) para atacar sistemas.
- Análisis de fallos para descubrir problemas en la red, tales como: ¿por qué el ordenador A no puede establecer una comunicación con el ordenador B?
- Medición del tráfico, mediante el cual es posible descubrir cuellos de botella en algún lugar de la red.
- Para los desarrolladores, en aplicaciones cliente-servidor. Les permite analizar la información real que se transmite por la red.
- Detección de intrusos, con el fin de descubrir crackers. Aunque para ello existen programas específicos llamados IDS (Intrusion Detection System, Sistema de Detección de intrusos), estos son prácticamente sniffers con funcionalidades específicas.
- Creación de registros de red, de modo que los crackers no puedan detectar que están siendo investigados.

Compilación e Instalación de MS2

Pasos a seguir, Primera Parte, Descomprimiendo y chequeando:

1. Crear una carpeta con el nombre mangosniffer (o el nombre que usted quiera).
2. Copiar el archivo mangoSniffer2.tar.gz en esa carpeta.
3. Descomprimir el archivo mangoSniffer2.tar.gz.
4. Al ejecutar un ls deben aparecer los siguientes archivos:
    - <main5.c> Código fuente revisión 5.
    - <ms2> Archivo binario del programa

    y la Siguiente carpeta

    - <libpcap-1.0.0> Carpeta que contiene todo el código fuente de la librería pcap utilizada por ms2.

Pasos a seguir, Segunda Parte, compilar e instalar librerías:

1. Desde la línea de comandos configurar-compilar-instalar la librería pcap. En la mayoría de los casos basta con entrar en la carpeta libpcap-1.0.0 (ej. cd libpcap-1.0.0) y luego hacer lo siguiente:
    - ./configure
    - make
    - sudo make install.

    En caso de presentarse algun error, infórmese con los archivos léeme que están dentro de la misma carpeta libpcap-1.0.0.

Pasos a seguir, Tercera Parte, compilar el programa MS2:

1. desde la línea de comandos, ubicados en la carpeta que contiene el archivo fuente (main5.c) compilar con la siguiente orden:
    - gcc -o ms2 main5.c -lpcap

Utilizando MS2

Para utilizar MS2 se debe hacer con privilegios administrativos, la forma correcta seria:

- sudo ./ms2 “tcp” 10.

Ahora pasaremos a explicar que significa esos parámetros “tcp” y 10.

El programa MS2 acepta dos parámetros:

- Primer Parámetro: una cadena entre comillas con el filtro para “husmear la red”.
- Segundo Parámetro: El numero de tramas a capturar.

Filtros

Es lo mas importante que nos permite hacer el MS2, el uso de filtros. Un filtro es una expresión que va detrás de las opciones y que nos permite seleccionar los paquetes que estamos buscando. En ausencia de ésta, el MS2 volcará todo el tráfico que vea el adaptador de red seleccionado.

La expresión que se usa para definir el filtro tiene una serie de primitivas y tres posibles modificadores a las mismas. Esta expresión será verdadera o falsa y hará que se imprima o no el paquete de datos.

Los 3 modificadores posibles son:

* tipo. Puede ser host, net o port. Indican respectivamente una maquina, por ejemplo host 192.168.1.1 , una red completa, por ejemplo net 192.168, o un puerto concreto, por ejemplo port 22. Por defecto se asume el tipo host.

* dir. Especifica desde o hacia donde se va a mirar el flujo de datos. Tenemos src o dst y podemos combinarlos con or y and. Para el caso de de protocolos punto a punto podemos sustituir por inbound o outbound. Por ejemplo si queremos la dirección de destino 10.10.10.2 y la de origen 192.168.1.2, el filtro

serma dst 10.10.10.2 and src 192.168.1.2 . Si se quiere que sea la dirección destino 192.168.1.1 o la dirección origen 192.168.1.2, serma dst 192.168.1.1 or src 192.168.1.2. Pueden seguirse combinando con la ayuda de paréntesis o las palabras or y and. Si no existe se supone src or dst. Por supuesto, esto se puede combinar con los modificadores de tipo anteriores.

* proto. En este caso es el protocolo que queremos capturar. puede ser tcp,udp,ip,ether (en este caso captura tramas a nivel de enlace,arp (peticiones arp), rarp (peticiones reverse-arp),fddi(para redes FDDI, pero realmente el encapsulado es igual al ether). Hay otros niveles de enlace para redes Decnet y lat, pero dado su escaso uso, me remito a la página de manual del programa.

Siempre podemos combinar expresiones con ayuda de paréntesis. Ojo con el tema de los paréntesis en los shell de Unix, porque son metacaracteres interpretan.

A continuación se dan las primitivas que pueden usarse. Lo que aparece entre [y] es opcional, y el | significa "o". El resto se tiene que poner si queremos poner el filtro con el comportamiento.

* [dst|src] host maquina. Cierto si la dirección destino u origen del paquete es maquina lo cual puede ser una dirección IPv4 (o IPv6 si se ha compilado soporte para el mismo), o un nombre del DNS. Si queremos restringir a dirección destino podemos restringir con dst. Para dirección origen src.

Ejemplos:

- Capturar el trafico cuya IP origen sea 192.168.1.1

  sudo ./ms2 “src host 192.168.1.1”

- Capturar todo el tráfico cuya dirección origen o destino sea 192.168.1.2

  sudo ./ms2 “host 192.168.1.2”

* ether src|dst|host edir. Este filtro es cierto sm la dirección origen (src), la destino (dst) o el cualquiera de las dos(host) coincide con edir. Hacer notar que src,dst o host es obligatorio especificarlo.

Ejemplos:

- Capturar el tráfico con destino a la dirección ethernet 0:2:a5:ee:ec:10.

  sudo ./ms2 “ether dst 0:2:a5:ee:ec:10”

- Capturar el tráfico que vaya a la máquina cuya dirección MAC es 0:2:a5:ee:ec:10.

  sudo ./ms2 “ether host 0:2:a5:ee:ec:10”

* gateway maquina. Cierto en caso de que el paquete use maquina como router. maquina debe estar definida en /etc/ethers y /etc/hosts. Realmente los paquetes que cumplen con esa condición son aquellos

que tienen como dirección ethernet destino maquina, pero ni la dirección IP destino u origen es maquina.

* [dst|src] net red. Cierto en caso de que la red de la dirección destino, origen o ambas sea red. El parámetro red puede ser una dirección numérica (por ejemplo 192.168.1.0) o bien un nombre que se resuelve a dirección, en los Unix, con ayuda del /etc/networks. Decir que también se admite el clásico direccionamiento CIDR. Podemos especificar una máscara poniendo red como net red mad mascara o bien usar /, net red/bits. Hacer notar que el uso de net ... mask no es compatible con direcciones IPv6. Si queremos hacer referencia a la red destino usamos dst como prefijo. Para la red origen usamos dst.

Ejemplos:

- Capturar todo el tráfico cuya red destino sea 192.168.1.0.

```
sudo ./ms2 “dst net 192.168.1.0”
```

- Capturar todo el tráfico cuya red origen sea 192.168.1.0/28

```
sudo ./ms2 “src net 192.168.1.0 mask 255.255.255.240 ”
sudo ./ms2 “src net 192.168.1.0/28 ”
```

- Capturar todo el tráfico con origen o destino en la 10.0.0.0/24

```
sudo ./ms2 “net 10.0.0.0/24 ”
sudo ./ms2 “net 10.0.0.0 mask 255.255.255.0 ”
```

* [dst|src] port puerto. Cierto en caso de que el puerto (ya sea udp o tcp) coincida con puerto. Si no se especifica dst o src, será cierto tanto puerto origen como destino. Si queremos restringir a destino usamos dst y a origen usamos src. El puerto es un valor numérico entre 0-65535 o bien un nombre que en Unix se resuelve a través del /etc/services.

Ejemplos:

- Capturar todo el tráfico con destino al puerto 23

```
sudo ./ms2 “dst port 23 ”
```

- Capturar todo el tráfico con destino o origen puerto 80

```
sudo ./ms2 “port 23”
```

* less longitud. Cierto en caso de que el tamaño del paquete sea menor o igual longitud.

* greater longitud. Cierto en caso de que el tamaño del paquete sea mayor o igual que longitud.

* ip proto protocolo. En este caso escucha el protocolo que se le indique. El protocolo puede ser icmp, icmp6, igmp (internet group managent protocol), igrp (interior gateway routing protocol), pim (protocol independent multicast), ah (IP Authentication header), esp (encapsulating security payload), udp o tcp. En caso de usar icmp, udp o tcp hay que escapar el protocolo, poniendo un \, es decir, ip

proto \icmp. Ojo con ese carácter que también hay que escaparlo en los shells de Unix.

Por comodidad se disponen los alias tcp, udp e icmp que equivalen a ip proto tcp or ip6 proto tcp, etc.

Ejemplos:

- Capturar el todo los paquetes icmp

```
sudo ./ms2 “ip proto \\ip ”
```

(en Unix hay que escapar el \).

- Capturar todo el tráfico udp

```
sudo ./ms2 “ip proto \\udp ”
sudo ./ms2 “udp ”
```

(el alias es más cómodo)

* ip6 proto protocolo. Cierto si es un paquete de IPv6 con el protocolo.
* ip6 protochain protocolo. Es un número que en los Unix puede leerse en /etc/protocols. En este caso lo que se busca es que dentro de los diferentes cabeceras que puede tener un paquete IPv6 una de ellas sea el protocolo especificado.
* ip protochain protocolo. Igual que el caso anterior pero para IPv4.
* ether broadcast. Cierto si la trama capturada va dirigida hacia la dirección de difusión ethernet. La palabra ether es opcional.
* ip broadcast. Cierto si el paquete va dirigido a la dirección de difusión de IP. Esta dirección se comprueba si es todo 0 o 1, o bien se comprueba la dirección local de la subred.
* ether multicast. Cierto si la trama va dirigida a una dirección multicast ethernet.
* ip multicast. Cierto si el paquete va dirigido a una dirección multicast IP.
* ip6 multicast. Cierto si el paquete va dirigido a una dirección multicast IPv6.
* ether proto protocolo. Cierto si el protocolo que contiene la trama es de tipo protocolo Los protocolos son ip, ip6, arp, rarp, atalk, aarp, decnet, sca, lat, mopdl moprc e iso. Además estos nombres son identificadores que deben de ser escapados con \.

Sin embargo hay una serie de alias que hacen más cómodo la expresión en los filtros. Dichas expresiones son ip,ip6 ,arp,rarp,aarp, decnet e iso, siendo equivalentes a ether proto ip, ether proto ip6, etc.

Ejemplos:

- Capturar todo tráfico arp

```
sudo ./ms2 “-n ether proto \\arp ”
sudo ./ms2 “arp ”
```

(el alias es más cómodo)
o Capturar todo tráfico ip

sudo ./ms2 “-n ether proto \\ip ”
sudo ./ms2 “pi”

* vlan [vlanid]. Cierto si la trama capturada es un paquete 802.1Q VLAN. Hacer notar de que esto cambia el resto de la interpretación del paquete capturado, en especial los desplazamientos a partir de los cuales empiezan a decodificar los protocolos, ya que se asume que estamos capturando paquetes que viajan en tramas VLAN. Por último si está presente el parámetro vlanid, sólo se mostraran aquellos paquetes que vayan a la VLAN vlanid.

Combinando los filtros
Se pueden combinar las expresiones anteriores con los ayuda de los operadores not, and y or (corresponden a la negación, el y lógico y el o lógico, dando lugar a filtros más complejos. Podemos usar también los equivalentes del lenguaje C: !, && o ||.

Ejemplos:

* Capturar todo el tráfico Web (TCP port 80)

sudo ./ms2 “tcp and port 80”

* Capturar el todas las peticiones DNS

sudo ./ms2 “udp and dst port 53”

* Capturar el tráfico al puerto telnet o ssh

sudo ./ms2 “tcp and (port 22 or port 23)”

* Capturar todo el tráfico excepto el web

sudo ./ms2 “tcp and not port 80”

Filtros Avanzados

Para los realmente muy cafeteros, el MS2 permite hacer filtros a mano, indicando que bytes de la trama queremos pillar y como los queremos interpretar. Cuando queremos definir filtros de esta manera la expresión general es:

expr relop expr

Donde relop puede ser cualquiera de las operaciones de relación de C: >,<, >= <=, = y !=. expr es una

expresión aritmética compuesta por una serie de números enteros, los operadores binarios de C, (+, -, *, /, & y |), un operador de longitud, len, y una serie de palabras reservadas que nos permiten el acceso a los diferentes paquetes de datos (ether, fddi, tr, ip, arp, rarp, tcp, udp, icmp e ip6).

Para acceder a los datos dentro de un paquete, usamos los modificadores anteriores y una expresión entera. Opcionalmente podemos especificar el tamaño de los datos que accedemos.

proto [expr : tam]

Asm por ejemplo, el primer byte de la trama ethernet será ether[0], la primera palabra será ether[0:2]. El parámetro tam puede ser 1 (por defecto y no hace falta especificarlo), 2 o 4. Por ejemplo,

Una nota, al menos en la página de manual de la versión 3.6.2 hacen notar que cuando se especifica tcp, udp u otro protocolo de nivel superior se hace referencia a IPv4. Esta limitación sin embargo no aparece en OpenBSD.

Otra nota a tener en cuenta: en caso de usar tcp[indice] o udp[índice], implícitamente se aplica una regla para averiguar si es un paquete fragmentado, es decir, usando la notación de estos filtros ip[0:2] & 0x1fff = 0. udp[0] o tcp[0] se refieren al primer byte de la cabecera UDP o TCP.

Analizando la Salida de MS2

$ sudo ./ms2 “tcp” 2

La salida sería la siguiente:

En primer lugar se obtiene la leyenda, la cual tiene el siguiente aspecto

Mango Computer c.a
Programado por Jose Andres Morales email:comprasmangocomputer@gmail.com
Mango Sniffer 2 V1.0 - MS2
Este programa es software libre

$ sudo <'>> NOMBRE_ARCHIVO'> ./ms2 <FILTRO><MAX PAQUETES A CAPTURAR>
Nomenclatura:
TN: Numero de Trama

Cabedera IP
ttl: Tiempo de Vida
TS: Tipo se Servicio
ID:ID del programa
OFF: Offset
CS: CheckSum
LT:Largo de Toda la Trama

Cabecera ARP
TH:Tipo de Hardware

TP:Tipo de Protocolo
LDH:Largo Direccion de Hardware
LDR:Largo Direccion de Red
CO:Codigo Operacion
HO:MAC del Hardware de Origen
HD:MAC del Hardware Destino

Paquete ICMP
TP:Tipo
CO:Codigo
CS:CheckSum
ID:ID del Programa
SQ: Secuencia
GW:Gateway
Paquete TCP
PO:Puerto Origen
PD:Puerto Destino
LC:Largo Cabecera TCP
NS:Numero de Secuencia
NA:Numero de Acuse de Recibo NACK
WI: Window Tamaño de la ventana del buffer
CS:CheckSum
UR:Puntero Urgente

Paquete UDP
PO:Puerto Origen
PD:Puerto Destino
CS:CheckSum

A Continuación se muestra por pantalla el inicio del programa:

Iniciando...

Obteniendo Tarjeta de red [OK]
-> eth0
Obtener red y mascara [OK]
-> Red: 192.168.2.0
-> Mascara de Red: 255.255.255.0
**** Opcion de Filtrado Activa ****
Compilar Filtro [OK] tcp
Estableciendo Filtro [OK]

Por último el momento que tanto hemos esperado, el programa pasa a mostrar la información de las dos tramas capturadas:

TN 1
0:15:58:af:f7:23->0:e0:4d:86:be:d IP 192.168.2.170->192.168.2.3 ttl:128 TS:0 ID:33826 OFF:0 CS:12464 LT:54
TCP PO:1038 PD:139 LC:20 NS:44246 NA:60176 cwr:0 ece:0 urg:0 ack:1 psh:0 rst:0 syn:0 fin:0 WI:65535 CS:1176 UR:0

TN 2
0:15:58:af:f7:23->0:e0:4d:86:be:d IP 192.168.2.170->192.168.2.3 ttl:128 TS:0 ID:33827 OFF:0 CS:12400 LT:117

TCP PO:1038 PD:139 LC:20 NS:44246 NA:60176 cwr:0 ece:0 urg:0 ack:1 psh:1 rst:0 syn:0 fin:0 WI:65535 CS:28421 UR:0

```
0000: 00 00 00 3B FF 53 4D 42 2E 00 00 00 00 18 07 C8   ...;.SMB........
0010: 00 00 00 00 00 00 00 00 00 00 00 00 02 00 FF FE   ................
0020: 00 00 C2 10 0C FF 00 DE DE EE 13 C0 1C 21 00 A0   .............!..
0030: 0F A0 0F 00 00 00 00 A0 0F 00 00 00 00 00 00      ...............
```

Pasemos analizar con detenimiento la información que nos está mostrando.

0:15:58:af:f7:23->0:e0:4d:86:be:d
MAC de origen y MAC de destino

IP 192.168.2.170->192.168.2.3
Es un paquete IP, donde IP origen es 192.168.2.170 y la IP destino es 192.168.2.3

ttl:128 TS:0 ID:33827 ... y demás información está muy bien documentada en la leyenda del programa.

Grabando la información a un archivo

Esta opción es muy importante, ya que permite que el usuario pueda estudiar con calma todas las tramas capturadas por el sistema.

Para enviar la información del ms2 a un archivo se debe hacer lo siguiente:

- $ sudo > peticiones_arp.txt ./ms2 “arp” 100

Nótese esto (> peticiones_arp.txt), donde peticiones_arp.txt es nombre del archivo a crear para guardar la información, en caso de existir lo borra y crea uno nuevo.
En caso de querer “agregar” información al archivo y no truncarlo, utilizar dos mayor que “>>” en vez de uno “>”, seria como sigue a continuación:

- $ sudo >> peticiones_arp.txt ./ms2 “arp” 100

Código Fuente del MS2 Revisión 5

```
/*
 * mangovisiblenet 1.0.0
 * Copyright (C) 2009 Mango Computer c.a Jose Andres Morales
 *
 * Email comprasmangocomputer@gmail.com for information about contributors
 *
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * compilar gcc -o ms2 main5.c -lpcap
 * Es necesario tener instalada la libreria pcap http://www.tcpdump.org/pcap.htm
 */



#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <pcap.h>
#include <netinet/if_ether.h>
#include <netinet/ether.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>
#include <netinet/udp.h>
#include <netinet/ip_icmp.h>
#include <net/if_arp.h>
#include <string.h>

#define PACKETSIZE      64
#define VERSION         "Version 1.0.0"
```

```
void dump(void* b, int len);
void llegada(u_char *useless,const struct pcap_pkthdr* pkthdr,const u_char* packet);
void uso();

struct packet_icmp
{
        struct icmphdr hdr;
        char msg[PACKETSIZE-sizeof(struct icmphdr)];
};



int main(int nump, char* param[]){


        char *net;                              // direccion de red
        char *mask;                             // mascara de subred
        char *dev;                              // nombre del dispositivo de red
        int ret;                                // codigo de retorno
        char errbuf[PCAP_ERRBUF_SIZE]; // buffer para mensajes de error
        bpf_u_int32 netp;                       // direcion de red en modo raw
        bpf_u_int32 maskp;                      // mascara de red en modo raw
        struct bpf_program fp;                  // contenedor con el programa compilado
        struct in_addr addr;
        pcap_t* descr;
        const u_char *packet;
        struct pcap_pkthdr hdr;
        struct ether_header *eptr; // Ethernet

        uso();

        printf("Obteniendo Tarjeta de red [");

        if ((dev = pcap_lookupdev(errbuf))==NULL){ //obtener la tarjeta de red
                printf("Fallo]-> %s\n", errbuf);
                exit(-1);
        }

        printf("OK] \n->       %s\n",dev);

        printf("Obtener red y mascara [");

        if ((ret = pcap_lookupnet(dev, &netp, &maskp, errbuf))==-1){
                printf("Fallo]-> %s\n", errbuf);
```

```
        exit(-1);
}

addr.s_addr = netp;
if ((net = inet_ntoa(addr))==NULL){
        printf("Fallo]->  Red -> inet_ntoa\n");
        exit(-1);
}

printf("OK] \n-> Red: %s\n",net);

addr.s_addr = maskp;
if ((mask = inet_ntoa(addr))==NULL){
        printf("[Fallo]->  Mascara de Red -> inet_ntoa\n");
        exit(-1);
}

printf("-> Mascara de Red: %s\n", mask);

if ((descr=pcap_open_live(dev,BUFSIZ,1,20,errbuf))==NULL){
        printf("Fallo en open live: %s\n",errbuf);
        exit(-1);
}

if (nump>1 ){
        printf("*** Opcion de Filtrado Activa ***\n Compilar Filtro [");

        if ((pcap_compile(descr, &fp, param[1], 0, netp)) == -1){
                printf("Fallo] compilando %s\n",param[1]);
                exit(-1);
        }
        printf("OK] %s\n",param[1]);

        printf("Estableciendo Filtro [");

        if ((pcap_setfilter(descr,&fp))==-1){
                printf("Fallo] estableciendo filtro \n");
                exit(-1);
        }

        printf("OK]\n");
        pcap_loop(descr,(nump==3 ? atoi(param[2]):1),llegada,NULL);

} else {
```

```
                pcap_loop(descr,2,llegada,NULL);
        }

        printf("\n..\n");

        exit(0);

}//end main


void dump(void* b, int len)
{   unsigned char *buf = b;
    int i, cnt=0;
    char str[17];
    memset(str, 0, 17);


    for ( i = 0; i < len; i++ )
    {
        if ( cnt % 16 == 0 )
        {
            printf("  %s\n%04X: ", str, cnt);
            memset(str, 0, 17);
        }
        if ( buf[cnt] < ' '  ||  buf[cnt] >= 127 )
            str[cnt%16] = '.';
        else
            str[cnt%16] = buf[cnt];
            printf("%02X ", buf[cnt++]);
    }

    printf("  %*s\n\n", 16+(16-len%16)*2, str);
}


/*
* Función que es llamada por la libreria pcap cada vez que llega
* una trama a la tarjeta de red
*
* void llegada(u_char *useless,const struct pcap_pkthdr* pkthdr,const u_char* packet){
*
* Donde el parametro const u_char* packet es una arreglo de bytes sin signo de los datos
* totales de la trama.
*/
```

```
void llegada(u_char *useless,const struct pcap_pkthdr* pkthdr,const u_char* packet){
        static int count;
        int largoCabeceras;
        struct ether_header *eptr;
        struct ip *ipc;
        struct ether_arp *arpc;
        int largoTrama;

        count++;
        printf("TN %d\n",count);                //Trama numero
        eptr = (struct ether_header *) packet;

        printf("%s->", ether_ntoa((struct ether_addr*)eptr->ether_shost));          //MAC origen
        printf("%s ", ether_ntoa((struct ether_addr*)eptr->ether_dhost));           //MAC destino

        if (ntohs(eptr->ether_type)==ETHERTYPE_IP){
                printf("IP ");
                ipc = (struct ip *) (packet+sizeof(struct ether_header));
                largoTrama = ntohs(ipc->ip_len)+sizeof(struct ether_header);
                printf("%s->",inet_ntoa(ipc->ip_src));                  //ip origen
                printf("%s ",inet_ntoa(ipc->ip_dst));
                printf("TTL:%d ",ipc->ip_ttl);
                printf("TS:%d ",ipc->ip_tos);                           //Tipo de Servicio
                printf("ID:%d ",ntohs(ipc->ip_id));
                printf("OFF:%d ",ntohs(ipc->ip_off));
                printf("CS:%d ",ntohs(ipc->ip_sum));                    //Check Suma
                printf("LT:%d\n",largoTrama);                           //len trama

        } else if (ntohs(eptr->ether_type)==ETHERTYPE_ARP){
                printf(" ARP ");
                arpc = (struct ether_arp *) (packet+sizeof(struct ether_header));
                printf("%d.%d.%d.%d->",arpc->arp_spa[0],arpc->arp_spa[1],arpc->arp_spa[2],arpc-
>arp_spa[3] );//ip origen
                printf("%d.%d.%d.%d\n",arpc->arp_tpa[0],arpc->arp_tpa[1],arpc->arp_tpa[2],arpc-
>arp_tpa[3] ); //ip destino
                printf("TH:%d",ntohs(arpc->ea_hdr.ar_hrd));         //Tipo de Hardware
                printf("TP:%d ",ntohs(arpc->ea_hdr.ar_pro));        //Tipo Protocolo
                printf("LDH:%d ",arpc->ea_hdr.ar_hln);              //Len Direccion Hardware
                printf("LDR:%d ",arpc->ea_hdr.ar_pln);              //Len Direccion Red
                printf("CO:%d ",ntohs(arpc->ea_hdr.ar_op));         //Codigo Operacion
                printf("HO:%s ", ether_ntoa((struct ether_addr*)arpc->arp_sha));
        //Hardware origen
                printf("HD:%s\n", ether_ntoa((struct ether_addr*)arpc->arp_tha));
```

```
        //Hardware Destino
                dump((void*)(packet+sizeof(struct ether_header)),sizeof(struct ether_arp));
                return;
        } else if (ntohs(eptr->ether_type)==ETHERTYPE_REVARP){
                printf(" RARP\n");
                return;
        } else {
                printf("Es de Tipo Desconocido\n");
                dump((void*)packet,sizeof(struct ether_header));
                return;
        }// end if


        switch(ipc->ip_p) {

                case 1: {
                        printf(" ICMP ");
                        struct packet_icmp *cicmp = (struct packet_icmp*)(packet+sizeof(struct
ether_header) + (ipc->ip_hl*4));
                        printf("TP:%d ", cicmp->hdr.type);
                        printf("CO:%d ", cicmp->hdr.code);
                        printf("CS:%d ", ntohs(cicmp->hdr.checksum));
                        printf("ID:%d ", ntohs(cicmp->hdr.un.echo.id));
                        printf("SQ:%d ", ntohs(cicmp->hdr.un.echo.sequence));
                        printf("GW:%d\n", ntohs(cicmp->hdr.un.gateway));
                        largoCabeceras = (sizeof(struct ether_header) + (ipc->ip_hl*4) + sizeof(struct
icmphdr));
                        dump((void*)(packet+largoCabeceras),(largoTrama-largoCabeceras));
                        break;
                } case 6: {
                        printf(" TCP ");
                        struct tcphdr *tcpc = (struct tcphdr*)(packet+sizeof(struct ether_header) + (ipc-
>ip_hl*4));
                        printf("PO:%d ", ntohs(tcpc->source));          //Puerto de Origen
                        printf("PD:%d ", ntohs(tcpc->dest));            //Puerto de Destino
                        int tmp_len_tcphdr = tcpc->doff*4;
                        printf("LC:%d ", tmp_len_tcphdr);
                        printf("NS:%d ", ntohs(tcpc->seq));             //Numero de Secuencia
                        printf("NA:%d ", ntohs(tcpc->ack_seq));         //Numero de acuse de recibo
                        printf("cwr:%d ", tcpc->res2&0x01);
                        printf("ece:%d ", tcpc->res2&0x02);
                        printf("urg:%d ", tcpc->urg);
                        printf("ack:%d ", tcpc->ack);
                        printf("psh:%d ", tcpc->psh);
```

```
                        printf("rst:%d ", tcpc->rst);
                        printf("syn:%d ", tcpc->syn);
                        printf("fin:%d ", tcpc->fin);
                        printf("WI:%d ", tcpc->window);
                        printf("CS:%d ", tcpc->check);
                        printf("UR:%d\n", tcpc->urg_ptr);

                        if (tmp_len_tcphdr>20){
                                largoCabeceras = (sizeof(struct ether_header) + (ipc->ip_hl*4) + 20);
                                printf("Opciones len %d\n",tmp_len_tcphdr-20);
                        } else {
                                largoCabeceras = (sizeof(struct ether_header) + (ipc->ip_hl*4) +
tmp_len_tcphdr);
                        }
                        dump((void*)(packet+largoCabeceras),(largoTrama-largoCabeceras));
                        break;
                } case 17: {
                        printf(" UDP ");
                        struct udphdr *udpc= (struct udphdr*)(packet+sizeof(struct ether_header) + (ipc-
>ip_hl*4));
                        largoCabeceras = (sizeof(struct ether_header) + (ipc->ip_hl*4) + sizeof(struct
udphdr));
                        printf("PO:%d ", ntohs(udpc->source));          //Puerto de Origen:
                        printf("PD:%d ", ntohs(udpc->dest));            //Puerto de Destino
                        printf("CS:%d\n", ntohs(udpc->check));          //check suma
                        dump((void*)(packet+largoCabeceras),(largoTrama-largoCabeceras));
                        break;
                } default: {

                        printf("Otro Paquete #%d \n",ipc->ip_p);
                        dump((void*)packet,largoTrama);
                }
        }//end switch
        printf("\n");

}

void uso(){

        printf("Mango Computer c.a \nPrograma do por: Jose Andres Morales
email:comprasmangocomputer@gmail.com\nMango Visible Red %s\nEste programa es sofware
libre\n\n? sudo <\'>> NOMBRE_ARCHIVO\'> ./mangovisiblenet <FILTRO><MAX PAQUETES A
CAPTURAR>\nNomenclatura:\nTN: Numero de Trama\n\nCabedera IP\nTTL: Tiempo de Vida\nTS:
Tipo se Servicio\nID:ID del programa\nOFF: Offset\nCS: CheckSum\nLT:Largo de Toda la
```

```
Trama\n\nCabecera ARP\nTH:Tipo de Hardware\nTP:Tipo de Protocolo\nLDH:Largo Direccion de Hardware\nLDR:Largo Direccion de Red\nCO:Codigo Operacion\nHO:MAC del Hardware de Origen\nHD:MAC del Hardware Destino\n\nPaquete ICMP\nTP:Tipo\nCO:Codigo\nCS:CheckSum\nID:ID del Programa\nSQ: Secuencia\nGW:Gateway\nPaquete TCP\nPO:Puerto Origen\nPD:Puerto Destino\nLC:Largo Cabecera TCP\nNS:Numero de Secuencia\nNA:Numero de Acuse de Recibo NACK\nWI: Window Tamaño de la ventana del buffer\nCS:CheckSum\nUR:Puntero Urgente\n\nPaquete UDP\nPO:Puerto Origen\nPD:Puerto Destino\nCS:CheckSum\n\nIniciando...\n\n",VERSION);
}
```