Ayuda Práctica: Capa de Transporte, UDP

Nicolás Macia [*], Andres Barbieri [**], Matías Robles [***]

13 de mayo de 2017

[*] nmacia@cespi.unlp.edu.ar
[**] barbieri@cespi.unlp.edu.ar
[***] mrobles@info.unlp.edu.ar

1. UDP (User Datagram Protocol)

UDP (User Datagram Protocol) es un protocolo de la capa de transporte que ofrece un servicio no orientado a conexión. La unidad de datos que envía o recibe, PDU (Protocol Data Unit), es conocido con el nombre de datagrama UDP o simplemente datagrama, aunque a veces por pertenecer a la capa de transporte se lo suele llamar segmento. Las aplicaciones que requieran de una entrega fiable y orden de secuencias de datos deberían utilizar el Protocolo TCP, o hacer que este control recaiga en la aplicación. El protocolo UDP goza del mismo mecanismo de multiplexado utilizado por el protocolo TCP. En la figura 1 se muestra el PDU del protocolo UDP.
Este protocolo esta definido en el documento [RFC-768]. Los usos principales de este son el Servicio de Nombres de Internet (DNS), y aplicaciones de streaming de audio y vídeo, por ejemplo VoIP con RTP. Otros protocolos de aplicación que lo utilizan son NTP, TFTP y BOOTP/DHCP. El número del protocolo es 17 (0x11 en hexadecimal) cuando se utiliza encapsulado en el Protocolo de Internet (IP).

```
? grep udp /etc/protocols
udp 17 UDP # user datagram protocol
...
```

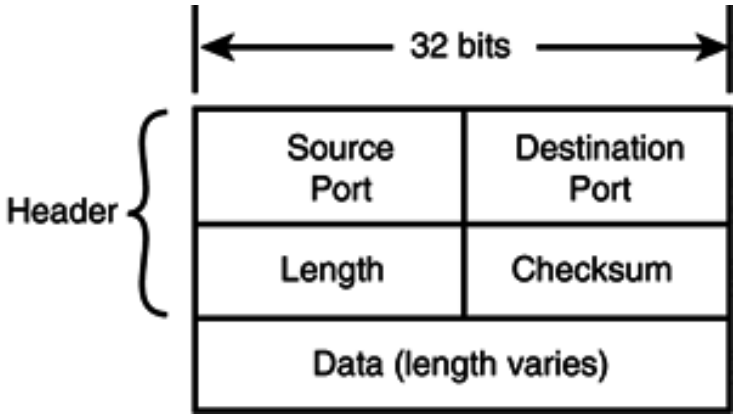

Figura 1: Datagrama UDP.

1.1. Ejemplos Sencillos con UDP

Para estos ejemplos se utilizará el comando Netcat `nc(1)` y se capturará y visualizará tráfico con las herramientas `tcpdump(8)`, `WIRESHARK(1)` o `ETHEREAL(1)` [COM05]. En las versiones más nuevas de netcat el switch `-p` es incompatible con `-l`, por lo que se debe indicar el comando sin esta opción.

1.1.1. Verificar los Procesos UDP

Primero se verifican los procesos utilizando comunicación UDP con el comando `netstat(8)`.

```
root@berlin:~# netstat -anup
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address  Stat   PID/Program name
udp        0      0 0.0.0.0:1024            0.0.0.0:*               4892/named
udp        0      0 10.20.1.100:53          0.0.0.0:*               4892/named
udp        0      0 172.20.1.100:53         0.0.0.0:*               4892/named
udp        0      0 127.0.0.1:53            0.0.0.0:*               4892/named
udp6       0      0 :::1025                 :::*                    4892/named
```

1.1.2. Levantar Proceso UDP

Se levanta un servicio UDP, se chequea que este esperando "conexiones".

```
root@berlin:~# nc -l -p 11111 -u
```

```
root@berlin:~# netstat -anup
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address  State   PID/Program name
udp        0      0 0.0.0.0:1024            0.0.0.0:*               4892/named
udp        0      0 10.20.1.100:53          0.0.0.0:*               4892/named
udp        0      0 172.20.1.100:53         0.0.0.0:*               4892/named
udp        0      0 127.0.0.1:53            0.0.0.0:*               4892/named
udp        0      0 0.0.0.0:11111           0.0.0.0:*               5108/nc
udp6       0      0 :::1025                 :::*                    4892/named
```

Luego se generan "conexiones", se "tipean" datos de entrada y se recibe la salida de texto. Previo a esto se capturan los datagramas UDP.

```
root@darkshark:/# tcpdump -n -i tap0 -s 1500 -w 00-udp.pcap
```

```
andres@darkshark:~? nc -u 172.20.1.100 11111
```

```
root@berlin:~# netstat -anup | grep 11111
udp        0      0 172.20.1.1:41863    172.20.1.100:11111      ESTABLISHED 10222/nc
```

El estado ESTABLISHED indica que este programa solo recibirá, a partir de ahora, datos del origen del primer datagrama recibido. A continuación las pruebas, lo "tipeado" por el cliente se marca en letra minúscula.

```
andres@darkshark:~? nc -u 172.20.1.100 11111
hola,
HOLA COMO VA?,
todo bien,
OK,
saludos.
```

```
^C

root@berlin:~# nc -l -p 11111 -u
hola,
HOLA COMO VA?,
todo bien,
OK,
saludos.
CHAU,
```

1.2. Ejemplo de UDP e ICMP

En la prueba anterior se ve que el cliente corta la comunicación primero. Debido a que el protocolo no contempla ninguna señalización de esto, el proceso actuando como servidor no es notificado y al querer enviar el último mensaje: `CHAU,`, es alertado por ICMP que el puerto al cual envía el datagrama (puerto del cliente asignado dinámicamente: 41863) no esta más disponible (Ver figura 2).

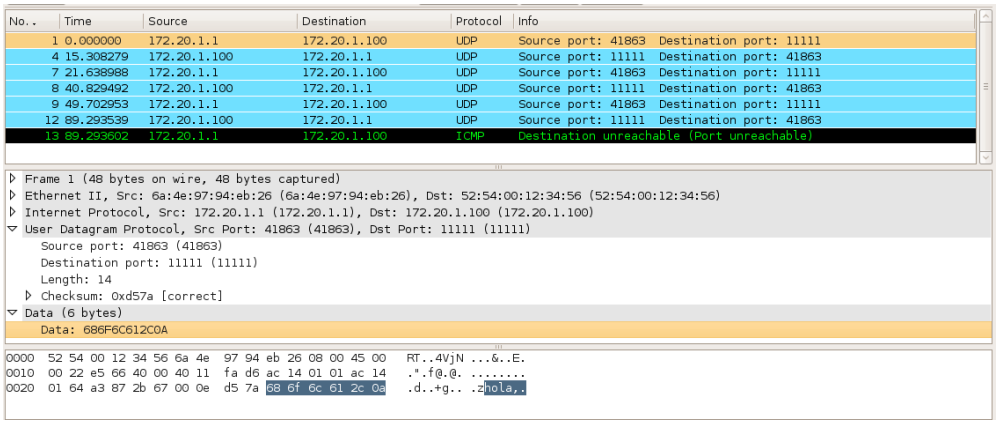

Figura 2: Captura de datagrama UDP.

El protocolo ICMP es utilizado como protocolo de señalización y control para notificar sobre la inexistencia de hosts, servicios o redes para IP y UDP. En particular en UDP se utiliza ICMP para señalizar condiciones anormales puesto que esta capa de transporte no tiene mecanismos propios para dicho propósito.

Otra forma de generarlo es directamente intentando enviar información a un puerto donde no hay un proceso en estado de espera de datos. En este caso lo "tipeado" por el cliente se escribe en letra MAYÚSCULA, y no hay respuesta del servidor.

```
root@darkshark:/# tcpdump -n -i tap0 -s 1500 -w 01-udp.pcap

andres@darkshark:~? nc -u -p 2222 172.20.1.100 12345
HOLA
andres@darkshark:~?
```

1.3. Ejemplo: UDP Connectionless

Debido a que UDP es un protocolo no orientado a conexión y en su implementación no mantiene ningún estado de la conexión, es posible repetir un proceso similar al primero en el cual se corta la comunicación, pero luego puede volver a restablecerse como si nada hubiese sucedido. Para esto es necesario engañar al programa Netcat `nc(1)`, ya que este al recibir la primera información, como servidor, luego cierra la espera desde cualquier origen y solo esperara recibir datos desde puerto cliente que envió el primer datagrama UDP, por esto es necesario enviar desde el mismo puerto e IP origen. Esto se debe a que en su programación, hace una llamada a la llamada de sistema (system call) `connect(2)` del API de sockets.

```
CONNECT(2)              Linux Programmer's Manual                    CONNECT(2)
...
If  the  socket  sockfd is of type SOCK_DGRAM  then  serv_addr  is  the address
to which datagrams are sent by default, and the  only address from   which
datagrams are received. If the socket is of type SOCK_STREAM or SOCK_SEQPACKET,
this call attempts to make a connection to the socket that is bound to the
address specified by serv_addr.
...
```

A continuación se muestra el ejemplo.

```
root@darkshark:/# tcpdump -n -i tap0 -s 1500 -w 02-udp.pcap

root@berlin:~# nc -l -p 11111 -u

root@berlin:~#  netstat -anup | grep 11111
udp        0      0 0.0.0.0:11111           0.0.0.0:*                       5133/nc


andres@darkshark:~? nc -u -p 2222 172.20.1.100 11111
hola desde 2222
^C

root@berlin:~# nc -l -p 11111 -u
hola desde 2222

root@berlin:~# netstat -anup | grep 11111
udp        0      0 172.20.1.100:11111      172.20.1.1:2222   ESTABLISHED 5133/nc


andres@darkshark:~?  nc -u -p 3333 172.20.1.100 11111
hola desde 3333
andres@darkshark:~?
```

```
root@berlin:~# nc -l -p 11111 -u
hola desde 2222

andres@darkshark:~? nc -u -p 2223 172.20.1.100 11111
hola desde 2223
andres@darkshark:~? nc -u -p 2222 172.20.1.100 11111
hola desde 2222 nuevamente
^C
```

En el servidor solo se ven los datos de los mensajes que llegaron desde el puerto 2222.

```
root@berlin:~# nc -l -p 11111 -u
hola desde 2222
hola desde 2222 nuevamente
```

Se puede ver los momentos donde se hace el `connect(2)`, primero del lado del cliente y luego del servidor si se hace un trace de las llamadas al sistema que se realizan con la herramienta `strace(1)`.

```
root@berlin:~# strace nc -l -p 11111 -u
...
brk(0x9b2c000)                          = 0x9b2c000
socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP) = 3
setsockopt(3, SOL_SOCKET, SO_REUSEADDR, [1], 4) = 0
bind(3, sa_family=AF_INET, sin_port=htons(11111), sin_addr=inet_addr("0.0.0.0"), 16) = 0
recvfrom(3,
```

Allí se queda esperando por datos de cualquoer cliente. Del lado del cliente veremos que llama a `connect(2)`.

```
andres@darkshark:~? strace nc -u -p 2222 172.20.1.100 11111
...
socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP) = 3
fcntl64(3, F_GETFL)                     = 0x2 (flags O_RDWR)
fcntl64(3, F_SETFL, O_RDWR|O_NONBLOCK)  = 0
connect(3, {sa_family=AF_INET, sin_port=htons(11111),
           sin_addr=inet_addr("172.20.1.100")}, 16) = 0
fcntl64(3, F_SETFL, O_RDWR)             = 0
poll([{fd=3, events=POLLIN}, {fd=0, events=POLLIN}], 2, -1
...
```

Luego si se envían datos desde el cliente

```
hola desde 2222
...
) = 1 ([{fd=0, revents=POLLIN}])
read(0, "hola de connect\n", 1024)      = 16
write(3, "hola de connect\n", 16)       = 16
```

Ahora el servidor hará el connect contra el equipo que envío primero los datos.

```
...
recvfrom(3,
           "hola de connect\n", 1024, MSG_PEEK,
           {sa_family=AF_INET, sin_port=htons(49156),
           sin_addr=inet_addr("10.0.1.10")}, [16]) = 16
connect(3, {sa_family=AF_INET, sin_port=htons(49156),
           sin_addr=inet_addr("172.20.1.1")}, 16) = 0
...
```

Para poder observar la capacidad de recibir información de diferentes fuentes podemos utilizar algunos de los servicios simples de TCP/IP proveídos por el super-daemon `inetd(8)`, desde cualquiera de sus variantes como bsd-inetd, xinetd, etc. En este caso se va a levantar el servicio de ECHO y el de DAYTIME, implementado dentro del mismo proceso.

```
root@darkshark:/# tcpdump -n -i tap0 -s 1500 -w 03-udp.pcap

root@berlin:~# cat /etc/inetd.conf

#
#       inetd internal services
#
daytime  dgram  udp nowait root internal
echo     dgram  udp nowait root internal


root@berlin:~# egrep "echo|daytime" /etc/services
echo 7/tcp
echo 7/udp
daytime 13/tcp
daytime 13/udp
...

root@berlin:~# /etc/init.d/inetutils-inetd restart

root@berlin:~# netstat -anup
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address        Foreign Address State  PID/Program name
udp        0      0 0.0.0.0:1024         0.0.0.0:*              4892/named
udp        0      0 10.20.1.100:53       0.0.0.0:*              4892/named
udp        0      0 172.20.1.100:53      0.0.0.0:*              4892/named
udp        0      0 127.0.0.1:53         0.0.0.0:*              4892/named
udp6       0      0 :::1025              :::*                   4892/named
udp6       0      0 :::7                 :::*                   5266/inetutils-inet
```

```
udp6       0      0 :::13                   :::*                    5266/inetutils-inet
```

Luego se generan datagramas UDP con el programa Netcat.

```
andres@darkshark:~? nc -u -p 2222 172.20.1.100 7
hola desde 2222
hola desde 2222

andres@darkshark:~? nc -u -p 3333 172.20.1.100 7
hola desde 3333
hola desde 3333
```

Se puede observar que el servidor de ECHO recibe datos desde cualquier cliente.

```
root@berlin:~# netstat -aunp | grep "udp6" | egrep "7|13"
udp6       0      0 :::7                    :::*                    5266/inetutils-inet
udp6       0      0 :::13                   :::*                    5266/inetutils-inet


root@darkshark:/home/andres# netstat -anup | egrep "2222|3333"
udp        0      0 172.20.1.1:3333         172.20.1.100:7          ESTABLISHED 22715/nc
udp        0      0 172.20.1.1:2222         172.20.1.100:7          ESTABLISHED 22714/nc
```

De forma similar sucede con el servidor de DAYTIME.

```
^C
andres@darkshark:~? nc -u -p 2222 172.20.1.100 13
<ENTER>
Thu Apr  2 15:25:17 2009
^C
```

1.4. Ejemplo de Fragmentación UDP

El tamaño máximo posible del contenido de un datagrama UDP, sería teóricamente el payload ofrecido por IP, lo que da 64K-1: 65535 bytes - (IP Header (20 bytes default) + UDP Header (8 bytes)), dando 65507 bytes. En [RFC-768] no se menciona nada acerca de este límite. En las implementaciones de UDP en general se utilizan valores más chicos, los habituales son 8K, 4K o 2K. Estos valores según [StevI] se derivan de tamaños de bloques utilizables por el sistema de archivos de red NFS (Network File System) desarrollado por Sun [1]. Otras implementaciones tratan un valor default reducido a menor que el MTU que se genera sobre Ethernet, por ejemplo 512 bytes, o 1500 - 28 = 1472 bytes. Este valor máximo habitualmente se puede cambiar desde la API de sockets con la llamada `setsockopt(2)`. Para ver como se fragmenta automáticamente por la aplicación se puede correr el siguiente ejemplo.

[1] El protocolo UDP en general no era utilizado por aplicaciones que transfieren muchos datos y parece que uno de los pocos casos de aplicaciones que hacían esto y a su vez uno de los protocolos más utilizados era el NFS.

```
root@darkshark:/# tcpdump -n -i tap0 -s 1500 -w 04-udp.pcap

root@berlin:~# nc -l -p 11111 -u > /dev/null

andres@darkshark:~? cat /dev/zero | nc -u -p 2222 172.20.1.100 11111
^C
```

Para generar datos podría utilizarse cualquiera de estos otros comandos alternativos.

```
andres@darkshark:~? cat /dev/urandom | nc -u -p 2222 172.20.1.100 11111

andres@darkshark:~? du -hs /var/log/daemon.log.0
336K /var/log/daemon.log.0

andres@darkshark:~? cat /var/log/daemon.log.0 | nc -u -p 2222 172.20.1.100 11111
```

En el caso de Linux al enviar datos en una sola llamada a `sendto(2)`, en la cual se especifica un valor mayor del permitido, se observa lo siguiente.

```
root@darkshark:/# tcpdump -n -i tap0 -s 1500 -w 06-udp.pcap

andres@darkshark:~? ./udp-peer -S 172.20.1.100 -s 65508
ERROR: at send data: Message too long

andres@darkshark:~? ./udp-peer -S 172.20.1.100 -s 65507
```

1.5. Ejemplo de Análisis de Campos UDP

Los campos UDP de acuerdo a la figura 1 son 4. Los 2 puertos origen y destino, la longitud y el checksum. Los puertos son valores de 16 bits, de los cuales el valor origen según [RFC-768] es opcional.

1.5.1. Ejemplo de SRC Port Opcional

Para esto se requiere alguna aplicación que permita configurarlo a 0 (cero), y como consecuencia privilegios de super-usuario. Se utiliza la aplicación `nmap(1)`.

```
root@darkshark:/# tcpdump -n -i tap0 -s 1500 -w 05-udp.pcap

root@darkshark:~# nc -u -p 0 172.20.1.100 7
invalid local port 0

root@darkshark:~# nmap --source-port 0 172.20.1.100 -sU -p 7
WARNING: a source port of zero may not work on all systems.

Starting Nmap 4.62 ( http://nmap.org ) at 2009-04-02 15:44 ART
```

```
Interesting ports on 172.20.1.100:
PORT  STATE         SERVICE
7/udp open|filtered echo
MAC Address: 52:54:00:12:34:56 (QEMU Virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 0.695 seconds
```

1.5.2. UDP Length

La longitud UDP es el total de todo el datagrama. El valor mínimo es 8, por lo que se deriva que es válido mandar un datagrama sin payload (vacío). Se puede generar un datagrama UDP sin datos con la herramienta `nmap(1)`.

```
root@darkshark:/# tcpdump -n -i tap0 -s 1500 -w 07-udp.pcap

root@darkshark:~# nmap --source-port 2222 172.20.1.100 -sU -p 7
WARNING: a source port of zero may not work on all systems.
...
```

1.5.3. UDP Checksum

El checksum UDP se calcula utilizando el mismo algoritmo de IPv4: Suma en Complemento a 1 (Ca1) tomando de a palabras de 16 bits y finalmente complementando el resultado, pero a diferencia de IPv4, este cubre el encabezado y los datos. Para el cálculo, tanto UDP como TCP incluyen datos adicionales llamados pseudo-header de 12 bytes. En el pseudo-header (ver figura 3) se incluyen campos del encabezado IPv4 (Direcciones y protocol) y se repiten campos del encabezado UDP (UDP length) además de un padding. El padding se hace en 0 (cero). Para el caso de UDP el checksum es opcional y se puede deshabilitar configurando su valor a todos 0 (ceros). En [RFC-768] se indica que el propósito de deshabilitarlo es para debugging o si existe el caso de algún protocolo de alto nivel que no lo necesite. Para deshabilitarlo se puede realizar desde la API de sockets con la llamada `setsockopt(2)` con el parámetro `SO_NO_CHECK`. Para IPv6 el checksum de UDP no es opcional de acuerdo a [RFC-2460] debido a la falta de este control en el protocolo de red.

1.6. UDP Broadcast y Multicast

En [RFC-919] se describe la capacidad de IPv4 de poder enviar mensajes de broadcast. A nivel de transporte el protocolo adecuado para enviar mensajes broadcast o multicast es UDP.

```
root@darkshark:/# tcpdump -n -i tap0 -s 1500 -w 08-udp.pcap

root@berlin:~# nc -u -b -p 3333 255.255.255.255 7 -z
```

1.7. Código UDP de Sockets

Analizar los siguientes códigos socket UDP. Tener en cuenta que están muy simplificados y faltos de todos chequeos.

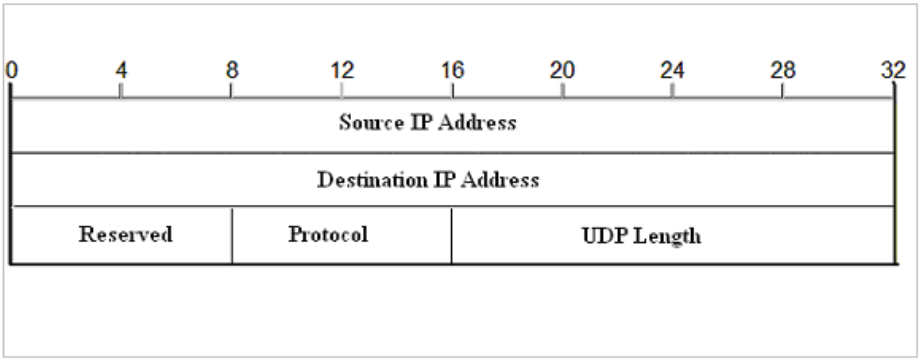

Figura 3: TCP y UDP Pseudo-header.

```
? cat udp-single-server.c
#include <unistd.h>                     /* close , read */
#include <sys/types.h>                  /* connect */
#include <sys/socket.h>                 /* socket , send , recv, ... */
#include <arpa/inet.h>                  /* ip socket , sockaddr_in */
#include <stdlib.h>                     /* exit , malloc , free  */
#include <errno.h>                      /* errno  */
#include <stdio.h>                      /* printf */

int main(int argc, char *argv[])
{
    int                 sock;
    struct sockaddr_in  addr;
    int                 addrlen;
    char                c;

    /* Crea el Socket */
    sock = socket(AF_INET, SOCK_DGRAM, 0);

    /* Construye y se asocia a una dirección local */
    addr.sin_family      = AF_INET;
    addr.sin_addr.s_addr = htonl(INADDR_ANY);
    addr.sin_port        = htons((uint16_t) 18000);

    bind(sock, (struct sockaddr *)&addr,
               (socklen_t) sizeof(struct sockaddr_in));

    /* Espera recibir un byte */
    recvfrom(sock, (char*) &c , sizeof(char), 0,
                   (struct sockaddr *) &addr,
                   (socklen_t*) &addrlen);
```

```
    /* Lo muestra en la STDOUT */
    printf("%c\n",c);

    close(sock);

    exit(0);
}
```

```
? cat udp-single-client.c
#include <unistd.h>                     /* close , read */
#include <sys/types.h>                  /* connect */
#include <sys/socket.h>                 /* socket , send , recv, ... */
#include <arpa/inet.h>                  /* ip socket , sockaddr_in */
#include <stdlib.h>                     /* exit , malloc , free  */
#include <errno.h>                      /* errno  */


int main(int argc, char *argv[])
{
    int                 sock;
    struct sockaddr_in  addr;
    char                c        =  '0';

    /* Crea el Socket */
    sock = socket(AF_INET, SOCK_DGRAM, 0);

    /* Construye una dirección remota a la cual mandar */
    addr.sin_family      = AF_INET;
    /* La IP remota se debe pasar como parámetro !!!! */
    addr.sin_addr.s_addr = inet_addr(argv[1]);
    addr.sin_port        = htons((uint16_t) 18000);

    /* Envia un byte */
    sendto(sock, (char*) &c , sizeof(char), 0,
                 (struct sockaddr *) &addr,
                 (socklen_t) sizeof(struct sockaddr));

    close(sock);

    exit(0);
}
```

Para probar se deben compilar y generar los dos ejecutables:

```
? gcc -Wall -o b udp-single-server.c
? gcc -Wall -o a udp-single-client.c
```

```
? b &

? netstat -anu | grep 18000
udp         0      0 0.0.0.0:18000           0.0.0.0:*

? ./a 127.0.0.1
```

2. Herramientas

2.1. Netcat

```
andres@darkshark:~? nc -h
[v1.10-38]
connect to somewhere:   nc [-options] hostname port[s] [ports] ...
listen for inbound:     nc -l -p port [-options] [hostname] [port]
options:
        -c shell commands       as `-e'; use /bin/sh to exec [dangerous!!]
        -e filename             program to exec after connect [dangerous!!]
        -b                      allow broadcasts
        -g gateway              source-routing hop point[s], up to 8
        -G num                  source-routing pointer: 4, 8, 12, ...
        -h                      this cruft
        -i secs                 delay interval for lines sent, ports scanned
        -k                      set keepalive option on socket
        -l                      listen mode, for inbound connects
        -n                      numeric-only IP addresses, no DNS
        -o file                 hex dump of traffic
        -p port                 local port number
        -r                      randomize local and remote ports
        -q secs                 quit after EOF on stdin and delay of secs
        -s addr                 local source address
        -T tos                  set Type Of Service
        -t                      answer TELNET negotiation
        -u                      UDP mode
        -v                      verbose [use twice to be more verbose]
        -w secs                 timeout for connects and final net reads
        -z                      zero-I/O mode [used for scanning]
port numbers can be individual or ranges: lo-hi [inclusive];
hyphens in port names must be backslash escaped (e.g. 'ftp\-data').
```

2.2. NMap

```
andres@darkshark:~? nmap -h
Nmap 4.62 ( http://nmap.org )
Usage: nmap [Scan Type(s)] [Options] {target specification}
TARGET SPECIFICATION:
  Can pass hostnames, IP addresses, networks, etc.
  Ex: scanme.nmap.org, microsoft.com/24, 192.168.0.1; 10.0.0-255.1-254
  -iL <inputfilename>: Input from list of hosts/networks
  -iR <num hosts>: Choose random targets
  --exclude <host1[,host2][,host3],...>: Exclude hosts/networks
  --excludefile <exclude_file>: Exclude list from file
HOST DISCOVERY:
  -sL: List Scan - simply list targets to scan
  -sP: Ping Scan - go no further than determining if host is online
```

```
  -PN: Treat all hosts as online -- skip host discovery
  -PS/PA/PU [portlist]: TCP SYN/ACK or UDP discovery to given ports
  -PE/PP/PM: ICMP echo, timestamp, and netmask request discovery probes
  -PO [protocol list]: IP Protocol Ping
  -n/-R: Never do DNS resolution/Always resolve [default: sometimes]
  --dns-servers <serv1[,serv2],...>: Specify custom DNS servers
  --system-dns: Use OS's DNS resolver
SCAN TECHNIQUES:
  -sS/sT/sA/sW/sM: TCP SYN/Connect()/ACK/Window/Maimon scans
  -sU: UDP Scan
  -sN/sF/sX: TCP Null, FIN, and Xmas scans
  --scanflags <flags>: Customize TCP scan flags
  -sI <zombie host[:probeport]>: Idle scan
  -sO: IP protocol scan
  -b <FTP relay host>: FTP bounce scan
  --traceroute: Trace hop path to each host
  --reason: Display the reason a port is in a particular state
PORT SPECIFICATION AND SCAN ORDER:
  -p <port ranges>: Only scan specified ports
    Ex: -p22; -p1-65535; -p U:53,111,137,T:21-25,80,139,8080
  -F: Fast mode - Scan fewer ports than the default scan
  -r: Scan ports consecutively - don't randomize
  --top-ports <number>: Scan <number> most common ports
  --port-ratio <ratio>: Scan ports more common than <ratio>
SERVICE/VERSION DETECTION:
  -sV: Probe open ports to determine service/version info
  --version-intensity <level>: Set from 0 (light) to 9 (try all probes)
  --version-light: Limit to most likely probes (intensity 2)
  --version-all: Try every single probe (intensity 9)
  --version-trace: Show detailed version scan activity (for debugging)
SCRIPT SCAN:
  -sC: equivalent to --script=safe,intrusive
  --script=<Lua scripts>: <Lua scripts> is a comma separated list of
           directories, script-files or script-categories
  --script-args=<n1=v1,[n2=v2,...]>: provide arguments to scripts
  --script-trace: Show all data sent and received
  --script-updatedb: Update the script database.
OS DETECTION:
  -O: Enable OS detection
  --osscan-limit: Limit OS detection to promising targets
  --osscan-guess: Guess OS more aggressively
TIMING AND PERFORMANCE:
  Options which take <time> are in milliseconds, unless you append 's'
  (seconds), 'm' (minutes), or 'h' (hours) to the value (e.g. 30m).
  -T[0-5]: Set timing template (higher is faster)
  --min-hostgroup/max-hostgroup <size>: Parallel host scan group sizes
  --min-parallelism/max-parallelism <time>: Probe parallelization
  --min-rtt-timeout/max-rtt-timeout/initial-rtt-timeout <time>: Specifies
      probe round trip time.
```

```
  --max-retries <tries>: Caps number of port scan probe retransmissions.
  --host-timeout <time>: Give up on target after this long
  --scan-delay/--max-scan-delay <time>: Adjust delay between probes
  --min-rate <number>: Send packets no slower than <number> per second
FIREWALL/IDS EVASION AND SPOOFING:
  -f; --mtu <val>: fragment packets (optionally w/given MTU)
  -D <decoy1,decoy2[,ME],...>: Cloak a scan with decoys
  -S <IP_Address>: Spoof source address
  -e <iface>: Use specified interface
  -g/--source-port <portnum>: Use given port number
  --data-length <num>: Append random data to sent packets
  --ip-options <options>: Send packets with specified ip options
  --ttl <val>: Set IP time-to-live field
  --spoof-mac <mac address/prefix/vendor name>: Spoof your MAC address
  --badsum: Send packets with a bogus TCP/UDP checksum
OUTPUT:
  -oN/-oX/-oS/-oG <file>: Output scan in normal, XML, s|<rIpt kIddi3,
     and Grepable format, respectively, to the given filename.
  -oA <basename>: Output in the three major formats at once
  -v: Increase verbosity level (use twice or more for greater effect)
  -d[level]: Set or increase debugging level (Up to 9 is meaningful)
  --open: Only show open (or possibly open) ports
  --packet-trace: Show all packets sent and received
  --iflist: Print host interfaces and routes (for debugging)
  --log-errors: Log errors/warnings to the normal-format output file
  --append-output: Append to rather than clobber specified output files
  --resume <filename>: Resume an aborted scan
  --stylesheet <path/URL>: XSL stylesheet to transform XML output to HTML
  --webxml: Reference stylesheet from Insecure.Org for more portable XML
  --no-stylesheet: Prevent associating of XSL stylesheet w/XML output
MISC:
  -6: Enable IPv6 scanning
  -A: Enables OS detection and Version detection, Script scanning and
      Traceroute
  --datadir <dirname>: Specify custom Nmap data file location
  --send-eth/--send-ip: Send using raw ethernet frames or IP packets
  --privileged: Assume that the user is fully privileged
  --unprivileged: Assume the user lacks raw socket privileges
  -V: Print version number
  -h: Print this help summary page.
EXAMPLES:
  nmap -v -A scanme.nmap.org
  nmap -v -sP 192.168.0.0/16 10.0.0.0/8
  nmap -v -iR 10000 -PN -p 80
SEE THE MAN PAGE FOR MANY MORE OPTIONS, DESCRIPTIONS, AND EXAMPLES
```

2.3. TCPDump

```
andres@darkshark:~? tcpdump --help
tcpdump version 3.9.8
libpcap version 0.9.8
Usage: tcpdump [-aAdDeflLnNOpqRStuUvxX] [-c count] [ -C file_size ]
                [ -E algo:secret ] [ -F file ] [ -i interface ] [ -M secret ]
                [ -r file ] [ -s snaplen ] [ -T type ] [ -w file ]
                [ -W filecount ] [ -y datalinktype ] [ -Z user ]
                [ expression ]
```

2.4. Hping

```
andres@darkshark:~? hping3 -h
usage: hping3 host [options]
  -h  --help      show this help
  -v  --version   show version
  -c  --count     packet count
  -i  --interval  wait (uX for X microseconds, for example -i u1000)
      --fast      alias for -i u10000 (10 packets for second)
      --faster    alias for -i u1000 (100 packets for second)
      --flood      sent packets as fast as possible. Don't show replies.
  -n  --numeric   numeric output
  -q  --quiet     quiet
  -I  --interface interface name (otherwise default routing interface)
  -V  --verbose   verbose mode
  -D  --debug     debugging info
  -z  --bind      bind ctrl+z to ttl           (default to dst port)
  -Z  --unbind    unbind ctrl+z
      --beep      beep for every matching packet received
Mode
  default mode     TCP
  -0  --rawip      RAW IP mode
  -1  --icmp       ICMP mode
  -2  --udp        UDP mode
  -8  --scan       SCAN mode.
                   Example: hping --scan 1-30,70-90 -S www.target.host
  -9  --listen     listen mode
IP
  -a  --spoof      spoof source address
  --rand-dest      random destionation address mode. see the man.
  --rand-source    random source address mode. see the man.
  -t  --ttl        ttl (default 64)
  -N  --id         id (default random)
  -W  --winid      use win* id byte ordering
  -r  --rel        relativize id field          (to estimate host traffic)
  -f  --frag       split packets in more frag.  (may pass weak acl)
  -x  --morefrag   set more fragments flag
  -y  --dontfrag   set dont fragment flag
  -g  --fragoff    set the fragment offset
```

```
  -m  --mtu         set virtual mtu, implies --frag if packet size > mtu
  -o  --tos         type of service (default 0x00), try --tos help
  -G  --rroute      includes RECORD_ROUTE option and display the route buffer
  --lsrr            loose source routing and record route
  --ssrr            strict source routing and record route
  -H  --ipproto     set the IP protocol field, only in RAW IP mode
ICMP
  -C  --icmptype    icmp type (default echo request)
  -K  --icmpcode    icmp code (default 0)
      --force-icmp  send all icmp types (default send only supported types)
      --icmp-gw     set gateway address for ICMP redirect (default 0.0.0.0)
      --icmp-ts     Alias for --icmp --icmptype 13 (ICMP timestamp)
      --icmp-addr   Alias for --icmp --icmptype 17 (ICMP address subnet mask)
      --icmp-help   display help for others icmp options
UDP/TCP
  -s  --baseport    base source port              (default random)
  -p  --destport    [+][+]<port> destination port(default 0) ctrl+z inc/dec
  -k  --keep        keep still source port
  -w  --win         winsize (default 64)
  -O  --tcpoff      set fake tcp data offset      (instead of tcphdrlen / 4)
  -Q  --seqnum      shows only tcp sequence number
  -b  --badcksum    (try to) send packets with a bad IP checksum
                    many systems will fix the IP checksum sending the packet
                    so you'll get bad UDP/TCP checksum instead.
  -M  --setseq      set TCP sequence number
  -L  --setack      set TCP ack
  -F  --fin         set FIN flag
  -S  --syn         set SYN flag
  -R  --rst         set RST flag
  -P  --push        set PUSH flag
  -A  --ack         set ACK flag
  -U  --urg         set URG flag
  -X  --xmas        set X unused flag (0x40)
  -Y  --ymas        set Y unused flag (0x80)
  --tcpexitcode     use last tcp->th_flags as exit code
  --tcp-timestamp   enable the TCP timestamp option to guess the HZ/uptime
Common
  -d  --data        data size                     (default is 0)
  -E  --file        data from file
  -e  --sign        add 'signature'
  -j  --dump        dump packets in hex
  -J  --print       dump printable characters
  -B  --safe        enable 'safe' protocol
  -u  --end         tell you when --file reached EOF and prevent rewind
  -T  --traceroute  traceroute mode               (implies --bind and --ttl 1)
  --tr-stop         Exit when receive the first not ICMP in traceroute mode
  --tr-keep-ttl     Keep the source TTL fixed, useful to monitor just one hop
  --tr-no-rtt        Don't calculate/show RTT information in traceroute mode
ARS packet description (new, unstable)
```

```
  --apd-send       Send the packet described with APD (see docs/APD.txt)
```

2.5. Wireshark

```
andres@darkshark:~? wireshark -h
Wireshark 1.0.3
Interactively dump and analyze network traffic.
See http://www.wireshark.org for more information.

Copyright 1998-2008 Gerald Combs <gerald@wireshark.org> and contributors.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Usage: wireshark [options] ... [ <infile> ]

Capture interface:
  -i <interface>           name or idx of interface (def: first non-loopback)
  -f <capture filter>      packet filter in libpcap filter syntax
  -s <snaplen>             packet snapshot length (def: 65535)
  -p                       don't capture in promiscuous mode
  -k                       start capturing immediately (def: do nothing)
  -Q                       quit Wireshark after capturing
  -S                       update packet display when new packets are captured
  -l                       turn on automatic scrolling while -S is in use
  -y <link type>           link layer type (def: first appropriate)
  -D                       print list of interfaces and exit
  -L                       print list of link-layer types of iface and exit

Capture stop conditions:
  -c <packet count>        stop after n packets (def: infinite)
  -a <autostop cond.> ...  duration:NUM - stop after NUM seconds
                           filesize:NUM - stop this file after NUM KB
                              files:NUM - stop after NUM files
Capture output:
  -b <ringbuffer opt.> ... duration:NUM - switch to next file after NUM secs
                           filesize:NUM - switch to next file after NUM KB
                              files:NUM - ringbuffer: replace after NUM files
Input file:
  -r <infile>              set the filename to read from (no pipes or stdin!)

Processing:
  -R <read filter>         packet filter in Wireshark display filter syntax
  -n                       disable all name resolutions (def: all enabled)
  -N <name resolve flags>  enable specific name resolution(s): "mntC"

User interface:
  -C <config profile>      start with specified configuration profile
  -g <packet number>       go to specified packet number after "-r"
```

```
  -m <font>                 set the font name used for most text
  -t ad|a|r|d|dd|e          output format of time stamps (def: r: rel. to first)
  -X <key>:<value>          eXtension options, see man page for details
  -z <statistics>           show various statistics, see man page for details

Output:
  -w <outfile|->            set the output filename (or '-' for stdout)

Miscellaneous:
  -h                        display this help and exit
  -v                        display version info and exit
  -P <key>:<path>           persconf:path - personal configuration files
                            persdata:path - personal data files
  -o <name>:<value> ...     override preference or recent setting
  --display=DISPLAY         X display to use
```

Referencias

[StevI] TCP/IP Illustrated, Volume 1: The Protocols, Addison-Wesley, 1994. W. Richard Stevens.

[Siever] Linux in a Nutshell, Fourth Edition June, 2003. O'Reilly. Ellen Siever, Stephen Figgins, Aaron Weber.

[RFC-768] http://www.rfc-editor.org/rfc/rfc768.txt User Datagram Protocol. (J. Postel 1980 ISI).

[RFC-919] http://www.rfc-editor.org/rfc/rfc919.txt. Broadcasting Internet Datagrams (Jeffrey Mogul 1984 Stanford).

[RFC-791] http://www.rfc-editor.org/rfc/rfc791.txt. Internet Protocol IP (Jon Postel 1981 USC-ISI IANA).

[RFC-792] http://www.rfc-editor.org/rfc/rfc792.txt. Internet Control Message Protocol ICMP (Jon Postel 1981 UNC-ISI California - IANA).

[RFC-2460] http://www.rfc-editor.org/rfc/rfc2460.txt. Internet Protocol, Version 6 (IPv6) Specification. (S. Deering, R. Hinden 1998).

[TCPADM] TCP/IP Network Administration By Craig Hunt. O'Reilly.

[LINUXIP] http://www.linux-ip.net/. Guide to IP Layer Network Administrator with Linux.

[COM05] Ethereal, Wireshark. Autor original Gerald Combs, 2005.
http://www.ethereal.com/.
http://www.wireshark.org/.

Índice

Índice de figuras