Sabuesos en la Red:

El escaneo de puertos

Autor:

Death Master

Índice de contenidos

Introducción

"Lara, no abras los puertos que te entran los barcos."

Mi amigo AcidBorg, en una clase aburrida...

Bienvenidos al quinto manual que publico.

Este texto supone un paso importante para mí en cuanto a la creación de manuales se refiere. Hasta ahora había escrito sobre software libre, sobre la seguridad informática en general, y sobre aspectos y técnicas concretas de ésta (a saber: criptografía y esteganografía). Pero este texto es distinto, pues es la primera vez que escribo sobre técnicas de hacking puro y duro.

El escaneo de puertos es seguramente la técnica de hacking más usada en el mundo, pues en cualquier "ataque" -bien se trate de una intrusión ilegal o de una auditoría legal- medianamente bien planificado, uno de los primeros pasos ha de ser obligatoriamente el escaneo metódico de los puertos de la máquina en cuestión. Lo malo es que la mayoría de la gente no sabe absolutamente nada sobre cómo funciona un escaneo de puertos: ellos saben que lo único que tienen que hacer es entrar en esa página tan *cool* con letras verdes sobre fondo negro y calaveras por doquier, bajar un programita actualizado por última vez en el 96, ejecutarlo e introducir la IP que quiere escanear en la caja de texto. El programa mágico hará el resto, y ellos no quieren saber cómo lo hace, porque no les importa.

Pero nosotros somos hackers. A nosotros nos interesa mucho más cómo se las apaña el programa para realizar semejante tarea que los resultados que pueda proporcionarnos. Y ahí es donde se distingue un hacker de cualquier otro tipo de persona: un hacker lee más que actúa, y cuando actúa lo hace con plena conciencia de sus actos.

Por otro lado, en contra de lo que mucha gente piensa, existen muchas técnicas distintas de escaneo de puertos. Cada una tiene sus ventajas y sus desventajas, y resulta sumamente interesante conocerlas para poder realizar el escaneo adecuado según la ocasión lo requiera.

Además, como buenos hackers, estamos interesados en tres aspectos de una técnica: cómo funciona, cómo llevarla a cabo correctamente y cómo plantear una defensa eficaz frente a ella. De todo eso hablaremos, y además de otras técnicas avanzadas que siempre resultan interesantes de conocer.

Por último, hoy en día no se concibe un texto sobre escaneo de puertos sin mencionar a **nmap**, el que es para mí hoy por hoy el mejor software existente para escaneo de puertos y otras técnicas como detección del fingerprint del sistema operativo. Decirme que escanee algo y os pediré una shell de Unix y **nmap**.

¿Todos los sabuesos listos? ¡A olfatear! Nos encanta meter las narices donde no nos llaman... 0:-)~

Death Master

Conceptos básicos

Para poder comprender cómo funciona un escaneo de puertos, es imprescindible conocer ciertos aspectos técnicos de la **arquitectura de Internet**. Esta "arquitectura de Internet" es la **familia de protocolos TCP/IP**: cuanto más profundamente conozcamos TCP/IP, más fácil será igualmente comprender todo lo relacionado con Internet, y este caso no es una excepción. Por desgracia, realizar un texto detallado sobre TCP/IP podría ocupar varias decenas de veces lo que ocupará este manual.

Existen infinidad de libros sobre TCP/IP en cualquier librería especializada, pero para leer estos libros hay que echarle muchas ganas, porque están escritos con un lenguaje tan técnico que casi todo el mundo acabará tirando la toalla y odiándome por haber mencionado siquiera TCP/IP. Pero no hay que desfallecer, pues existen textos muy buenos y fácilmente comprensibles que pueden ayudarnos a conocer TCP/IP de una forma más sencilla pero no por ello menos rigurosa. Personalmente, de los textos sobre el tema que he leído, creo que el mejor para todos aquellos que sepan poco o nada del tema es el realizado por el siempre genial **Vic_Thor** (le mando un saludo desde aquí :P). Podréis encontrar toda la información sobre su **Taller de TCP/IP** en: http://www.hackxcrack.com/phpBB2/viewtopic.php?t=10306.

Aún así, y aunque recomiendo encarecidamente el seguir el **Taller de TPC/IP de Vic_Thor**, yo voy a echar un somero vistazo a ciertos aspectos de TCP/IP que tienen especial importancia en todo lo relacionado con los escaneos de puertos.

Estableciendo conexiones: sockets, puertos e IP's

En la arquitectura TCP/IP, las conexiones se realizan mediante el establecimiento de **sockets**. ¿Y qué es un socket? Un **socket** es la **combinación de una máquina y un puerto con otra (o la misma) máquina y otro puerto**. ¿Y qué es un **puerto**? Un **puerto** es un **número de 16 bits** (comprendido entre 0 y 65535) que **permite establecer conexiones diferenciadas** entre máquinas. Los puertos desde el 0 al 1024 son puertos "reservados" para aplicaciones y protocolos conocidos, y los puertos del 1025 al 65535 son de uso libre. Esto es la teoría, claro, pues en la práctica nadie nos impide usar un puerto para lo que nos de la gana.

Un puerto puede tener **tres estados**: **abierto**, en cuyo caso **aceptará conexiones**; **cerrado**, en cuyo caso **rechazará cualquier intento de conexión**; y **bloqueado** o **silencioso**, en cuyo caso **ignorará cualquier intento de conexión**.

Una definición más casera de puerto sería los distintos "enchufes" que tenemos disponibles en un protocolo (**los puertos TCP y UDP son independientes**) para poder realizar conexiones.

Así pues, **toda conexión en Internet está identificada por un socket**, es decir, una máquina de origen con su puerto, y una máquina de destino con su puerto. En realidad está identificada por más cosas... pero ahora mismo no nos interesan.

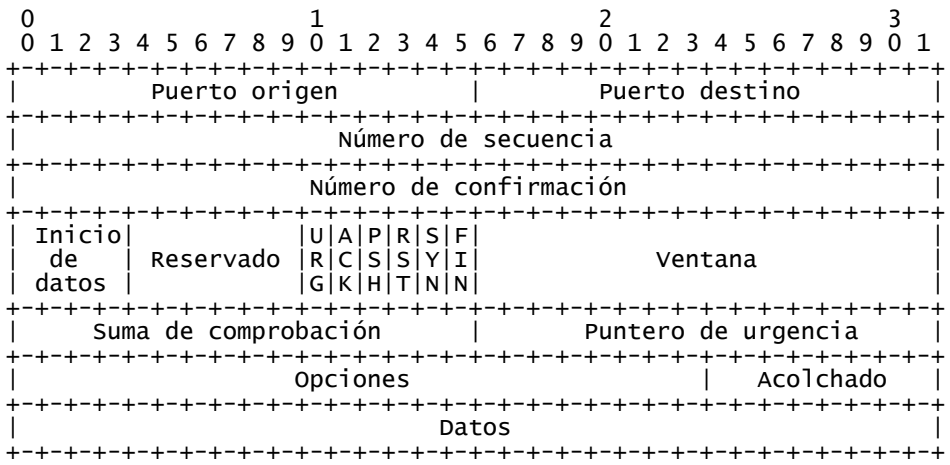
¿Y cómo identificamos una máquina en Internet? Por su **IP**. La **IP** (hablamos de IPv4) es una **dirección lógica de 32 bits** (cuatro números de 8 bits separados por puntos) que **identifica de forma unívoca a un host dentro de una red**. Así, podemos ampliar el anterior concepto de socket diciendo que se compone de un par de direcciones IP y puertos. Por ejemplo, ahora mismo acabo de realizar una consulta en http://www.google.es y consultando las conexiones de mi máquina encuentro una conexión TCP establecida entre 192.168.0.11:4191 y 66.102.11.99:80. Ahora algunos pensarán *"un momento, esa IP (192.168.0.11) la tengo yo en mi red, y acabas de decir que identificaba a una máquina de forma unívoca"*. Dije que identificaba de forma unívoca a un host **dentro de una red**, y **192.168.0.11 es una IP de mi red local**. La conexión real en la red de Internet se realiza entre mi IP pública y la IP de google, y mi router se encarga luego de enrutar Internet con mi red local.

Resumiendo...
Un **socket** es un conjunto **IP:puerto <> IP:puerto**.
Un **puerto** es un **número de 16 bits** que indica la **numeración lógica** de la conexión.
Una **IP** es una **dirección lógica de 32 bits** que **identifica a una máquina dentro de una red**.

Cabeceras de paquetes TCP, UDP e ICMP

Los protocolos más utilizados en Internet son (por este orden): TCP, UDP e ICMP. Aunque existen otros tipos de protocolos que también son muy importantes, estos tres serán con los que trabajemos a la hora de realizar escaneos de puertos. Por ello vamos a echar un vistazo a la estructura que tiene un paquete perteneciente a cada uno de ellos y vamos a realizar una pequeña explicación de sus campos y del protocolo en general.

TCP



RFC #793: ftp://ftp.rfc-editor.org/in-notes/rfc793.txt

TCP (Transmision Control Protocol) es sin duda el más complejo de los protocolos que vamos a tratar. La característica más importante de TCP es que se trata de un **protocolo orientado a conexión**, lo cual tiene tres consecuencias importantes: **crea conexiones virtuales a través de sockets** que permanecen activos el tiempo que dura la conexión; **los datos se envían ordenados** (lo cual no necesariamente significa que lleguen en orden...); y se realiza un constante **control de flujo** para evitar congestionar el ancho de banda disponible.

Ahora echemos un vistazo a los campos que componen el paquete TCP, teniendo en cuenta que los números en la parte superior del paquete representan un contador de bits.

Puerto de origen (16 bits): Se trata del **puerto origen** de la conexión.
Puerto de destino (16 bits): Se trata del **puerto destino** de la conexión.

Número de secuencia (32 bits): Este número es el que **identifica de forma unívoca a un paquete de datos dentro de una misma conexión**. El número de secuencia es el **orden en bytes** (siempre como un múltiplo de 32 bits) que ocupan los datos del propio paquete. Se obtiene sumando el tamaño del paquete inmediatamente anterior en la conexión al número de secuencia del mismo.

Número de confirmación (32 bits): Este número sirve para **confirmar la recepción de un paquete**. También representa el **orden en bytes** de los datos, y se calcula de igual forma que el número de secuencia, pero respecto a los paquetes recibidos.

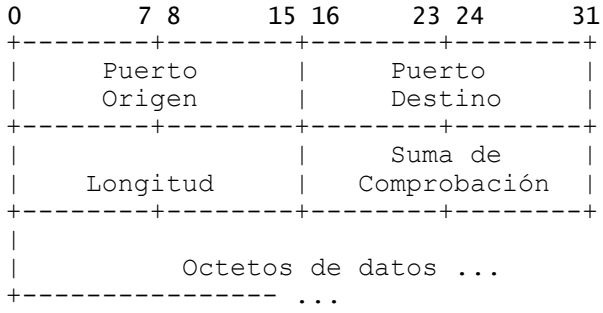
Inicio de datos (4 bits): Indica el **punto donde comienzan los datos** y acaba la cabecera TCP. Esto es debido a que existen campos opcionales que no siempre se incluyen. Su valor habitual es 5 (representa la información como **múltiplos de 32 bits**), pero puede llegar a ser 6.
Reservado (6 bits): Espacio reservado. Su valor es siempre 000000.
URG (1 bit), ACK (1 bit), PSH (1 bit), RST (1 bit), SYN (1 bit), FIN (1 bit): Se trata de las distintas **banderas (flags)** del protocolo TCP. Las trataremos más adelante en detalle.
Ventana (16 bits): Es el campo encargado del **control de flujo**. Indica el **número de bytes que podemos recibir en el próximo paquete**, y su finalidad es evitar la saturación de la conexión.

Suma de comprobación (16 bits): La **suma de comprobación (checksum)** es un número que permite **comprobar** que la cabecera de datos no ha llegado corrupta al destinatario. Se calcula mediante la **suma en complemento a uno de 16 bits** de una cabecera especial (**) que contiene los siguientes datos: dirección IP de origen (32 bits), dirección IP de destino (32 bits), campo reservado con valor 0 (8 bits), protocolo (8 bits) y tamaño del paquete (16 bits). Para saber más sobre la suma de comprobación es interesante leer el **RFC #1071** (ftp://ftp.rfc-editor.org/in-notes/rfc1071.txt).
Puntero de urgencia (16 bits): Para optimizar el envío de paquetes, **TCP permite combinar datos urgentes con datos normales**. El puntero de urgencia indica **a partir de qué byte** los datos dejan de ser urgentes para ser datos normales.

Opciones (24 bits): Este campo está destinado a contener **opciones especiales** poco comunes del protocolo TCP. A nosotros por ahora nos interesa poco. :-P
Acolchado (8 bits): Espacio reservado, su finalidad es completar los 24 bits del campo de opciones con otros 8 bits para lograr una cabecera múltiplo de 32 bits. Su valor es siempre 00000000.

Datos (x bits): Los datos del paquete propiamente dichos.

UDP



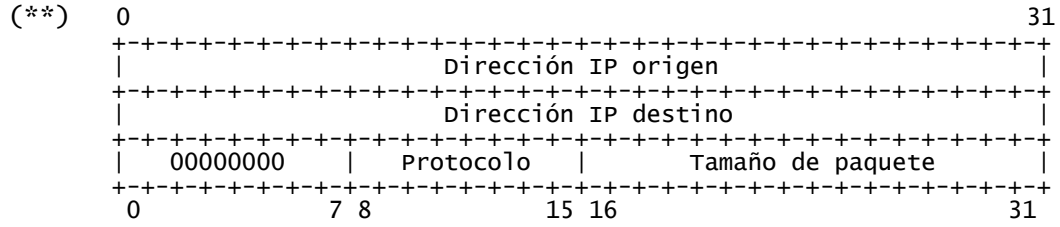
RFC #768: ftp://ftp.rfc-editor.org/in-notes/rfc768.txt

UDP (User Datagram Protocol) es un protocolo mucho más sencillo que TCP. Simplemente echando un vistazo a la composición de su cabecera podemos ver que ésta es también mucho más simple. UDP es un protocolo no orientado a conexión, lo cual tiene implicaciones más allá de la ausencia del establecimiento de conexiones virtuales o la ausencia de control de flujo. Más adelante veremos unas consideraciones sobre esto y estudiaremos más a fondo las diferencias entre TCP y UDP. Ahora echemos un ojo a la cabecera:

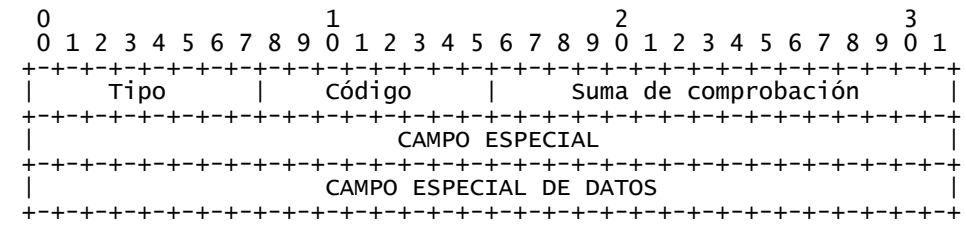
Puerto de origen (16 bits): Se trata del **puerto origen** de la conexión.
Puerto de destino (16 bits): Se trata del **puerto destino** de la conexión.

Longitud (16 bits): Es la **longitud del paquete UDP** expresada en bytes.
Suma de comprobación (16 bits): La **suma de comprobación (checksum)** es un número que permite **comprobar** que la cabecera de datos no ha llegado corrupta al destinatario. Se calcula mediante la **suma en complemento a uno de 16 bits** de una cabecera especial (**) que contiene los siguientes datos: dirección IP de origen (32 bits), dirección IP de destino (32 bits), campo reservado con valor 0 (8 bits), protocolo (8 bits) y tamaño del paquete (16 bits). Para saber más sobre la suma de comprobación es interesante leer el **RFC #1071** (ftp://ftp.rfc-editor.org/in-notes/rfc1071.txt).

Datos (x bits): Los datos del paquete propiamente dichos.



ICMP



RFC #792: ftp://ftp.rfc-editor.org/in-notes/rfc792.txt

ICMP (Internet Control Message Protocol) es un protocolo bastante **simple** que está orientado a **fines informativos o de control de errores**. Un paquete ICMP normalmente sirve para avisar al software de algún evento que requiere especial atención. Pero hay algo que hace a los paquetes ICMP algo más complejos, y es que excepto los primeros 32 bits de la cabecera, los demás campos **no son fijos** para todo paquete ICMP. Es más, una cabecera ICMP puede ocupar desde 96 bits hasta 160 bits **dependiendo del tipo de mensaje ICMP** que codifique ese paquete. Para comprender mejor esto, recomiendo en este caso más que nunca leerse el RFC, que en el caso de ICMP no es especialmente extenso ni complejo. Veamos la estructura de la cabecera ICMP:

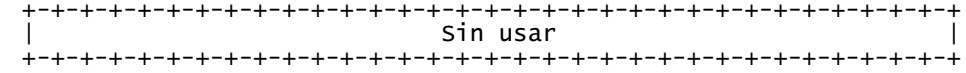
Tipo (8 bits): Indica el **tipo de mensaje ICMP:**

0=echo reply
3=destination unreachable
4=source quench
5=redirect
8=echo
11=time exceeded
12=parameter problem
13=timestamp
14=timestamp reply
15=information request
16=information reply

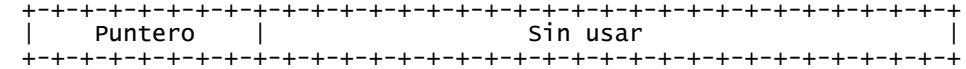
Código (8 bits): Indica, dentro de cada tipo de mensaje ICMP, una **opción concreta** de éste. Por ejemplo, en paquetes ICMP "time exceeded", un código de 0 significaría que se ha excedido el TTL (Time To Live) y un código de 1 que se ha excedido el tiempo de reensamblado de los fragmentos de un paquete.

Suma de comprobación (16 bits): La **suma de comprobación (checksum)** es un número que permite **comprobar** que la cabecera de datos no ha llegado corrupta al destinatario. Se calcula mediante la **suma en complemento a uno de 16 bits de la suma en complemento a uno del paquete ICMP** comenzando por el campo de tipo. Si es correcta, debe ser 0.

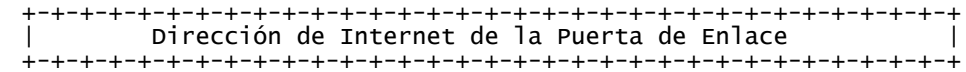
Campo especial (32 bits): Este campo puede variar en función del tipo de mensaje ICMP construido...



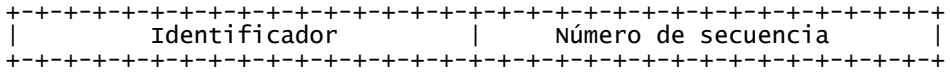
Tipo de ICMP: destination unreachable, time exceeded, source quench.



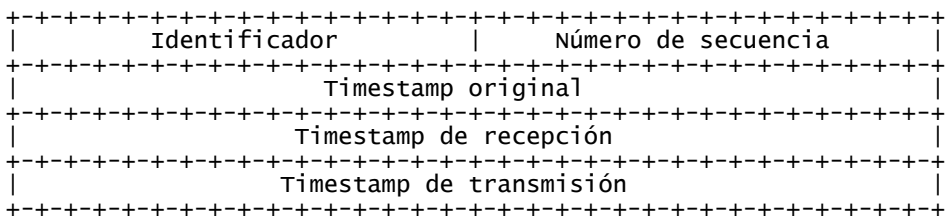
Tipo de ICMP: parameter problem.



Tipo de ICMP: redirect.

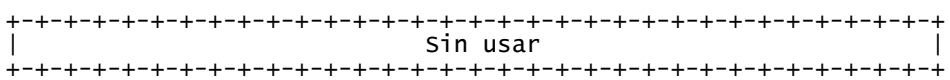


Tipo de ICMP: echo, echo reply, information request, information reply.

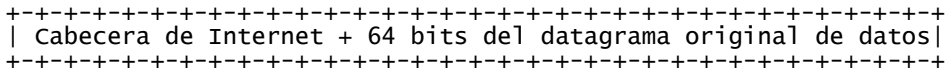


Tipo de ICMP: timestamp, timestamp reply.

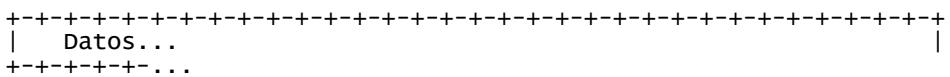
Campo especial de datos (32 bits): Este campo puede variar en función del tipo de mensaje ICMP construido...



Tipo de ICMP: timestamp, timestamp reply, information request, information reply.



Tipo de ICMP: destination unreachable, time exceeded, parameter problem, source quench, redirect.



Tipo de ICMP: echo, echo reply.

Flags TCP

Cuando vimos la cabecera TCP nombré las **banderas (flags)** del protocolo TCP. ¿Y qué son esas flags? Son 6 bits (**1 bit por cada flag**) de control. Cada flag es un **indicador especial** que cuando está activo (su valor es 1) provoca unos efectos distintos. Veamos cada uno en detalle:

Flag URG (Urgent)

Este flag indica que **el paquete contiene datos urgentes**. Como ya dijimos cuando tratamos la cabecera TCP, en un paquete pueden **combinarse datos urgentes con datos no urgentes**, usando en este caso el **puntero de urgencia** para marcar dónde terminan los datos urgentes y comienzan los normales.

Este flag no es muy usado, pero sí muy útil. Por ejemplo, si en una sesión de telnet pulsamos **control+C** para cortar el comando en ejecución (por ejemplo un dpkg -l en remoto :-P), ese dato se enviará como urgente y se tratará antes que los datos normales del paquete.

Flag ACK (Acknowledgement)

Este flag activo indica que además de los datos que pueda contener el paquete, éste sirve como **confirmación** de un paquete anteriormente recibido. Por tanto, para que el **número de confirmación** sea tomado en cuenta por la **pila TCP/IP**, el flag ACK debe estar activado.

Flag PSH (Push)

El flag PSH indica que **se debe vaciar el buffer** de transmisión o recepción (según se trate del emisor o el receptor). Cuando deseamos enviar una cantidad de información grande dividida en paquetes, éstos se van situando en un **buffer de transmisión FIFO (First In First Out)** hasta que el último de ellos está preparado. Este último paquete tiene activado **el flag push e indica que se debe vaciar el buffer** y comenzar el envío de paquetes.

Al llegar los datos al receptor, se van situando en otro **buffer FIFO de recepción**. Cuando llega el paquete con el flag push, los paquetes salen del buffer y pasan a la pila. No siempre los paquetes llegan en orden, por lo que puede que llegue el paquete con el flag push y falten aún algunos paquetes, pero esto no supone un problema porque se esperarán los paquetes ausentes y se reconstruirán luego todos gracias al **número de secuencia**.

A la hora de enviar datos es común **combinar el flag URG con el flag PUSH**, para evitar que los datos urgentes se retrasen en el buffer.

Flag RST (Reset)

Cuando enviamos un paquete con el flag RST activado, le estamos diciendo al otro extremo de la conexión que ha habido **algún tipo de problema** con la sincronización de la conexión (quizá números de secuencia o de confirmación incorrectos). Así, **el flag RST indica que la conexión ha de cerrarse y volverse a iniciar** para **sincronizar correctamente** ambas partes y continuar con lo que se estaba haciendo.

Flag SYN (Synchronization)

El flag SYN es usado cuando queremos indicar un intento de **nueva conexión** al otro host. El proceso concreto de establecimiento de nuevas conexiones lo veremos en detalle un poco más adelante.

Flag FIN (Finalization)

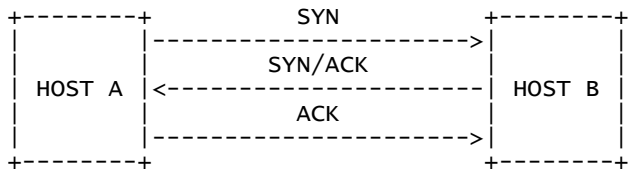
El flag FIN activo indica al otro host que **deseamos cerrar la conexión**, y quedamos a la espera de que el otro host también esté listo para cerrarla.

Aspectos importantes en conexiones TCP

Hay ciertos aspectos de las conexiones TCP que conviene conocer para entender mejor el funcionamiento de los escaneos de puertos. Veamos cuáles son:

Saludo en tres tiempos (Three way handshake)

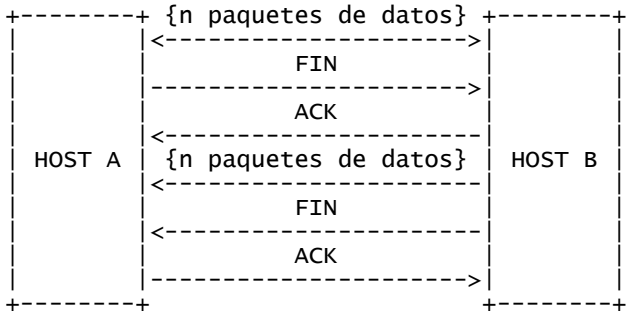
Como dije cuando hablábamos del flag SYN, el **establecimiento de conexiones en TCP** tiene un protocolo prefijado que ha de respetarse y que tiene por finalidad sincronizar los dos host que participan en la conexión. El **saludo en tres tiempos** se resume de la siguiente manera:



El HOST A envía un paquete con **el flag SYN levantado** para **sincronizar con el HOST B**, que a su vez responde con un paquete con **los flags SYN y ACK levantados**, como **confirmación** de recepción del paquete anterior y para **sincronizarse con el HOST A**. Por último, el HOST A **confirma la recepción** del paquete SYN/ACK mediante un paquete con el **flag ACK levantado**. A partir de ese momento **la conexión está establecida** y puede continuar el flujo de paquetes entre hosts.

Finalización de una conexión TCP

Al igual que la conexión TCP debe ser establecida de una forma correcta, la finalización (sin errores) de una conexión TCP también sigue unos pasos:



Tras una conexión con envío bidireccional de datos, el HOST A indica el deseo de **finalizar la conexión** mediante el envío de un paquete con el **flag FIN levantado**. El HOST B responde con un paquete con el **flag ACK levantado** para **confirmar la recepción** del paquete FIN, tras lo cual **termina de enviar los paquetes** que tuviera pendientes para dar por finalizada la conexión. Cuando todos son enviados, el HOST B envía un paquete con el **flag FIN levantado** al HOST A, que responde con un paquete con el **flag ACK levantado** para confirmar la recepción, y ambos consideran la **conexión finalizada**.

Estados TCP

Como ya dijimos, TCP es un protocolo orientado a conexión, y como tal tiene unos **estados definidos** según en qué punto de la conexión se encuentre el socket. Estos estados están detallados en el **RFC #793** y son:

LISTEN: Un servidor que espera conexiones de un cliente y **escucha un puerto**, genera un **socket con estado LISTEN**. Cuando el cliente se conecte, se creará un socket con la conexión establecida y otro que quede a la escucha.

SYN-SENT: Al enviar un paquete con el **flag SYN levantado**, como primer paso para el saludo en tres tiempos de una conexión, el socket entra en el **estado SYN-SENT**.

SYN-RECEIVED: Cuando tiene lugar el **segundo paso del saludo** en tres tiempos y se responde al primer SYN con un SYN/ACK, los sockets tienen **estado SYN-RECEIVED**.

ESTABLISHED: Una vez se **completa el saludo** en tres tiempos se entra en **estado ESTABLISHED** y se permanece en él durante todo el tiempo que dura la conexión.

FIN-WAIT-1: El socket entra en este estado una vez envía el paquete con el **flag FIN levantado** pero aún **no ha recibido la confirmación** de ese paquete. Solamente se reciben datos.

FIN-WAIT-2: Una vez recibida la **confirmación ACK del paquete FIN**, entramos en este estado. Solamente se reciben datos.

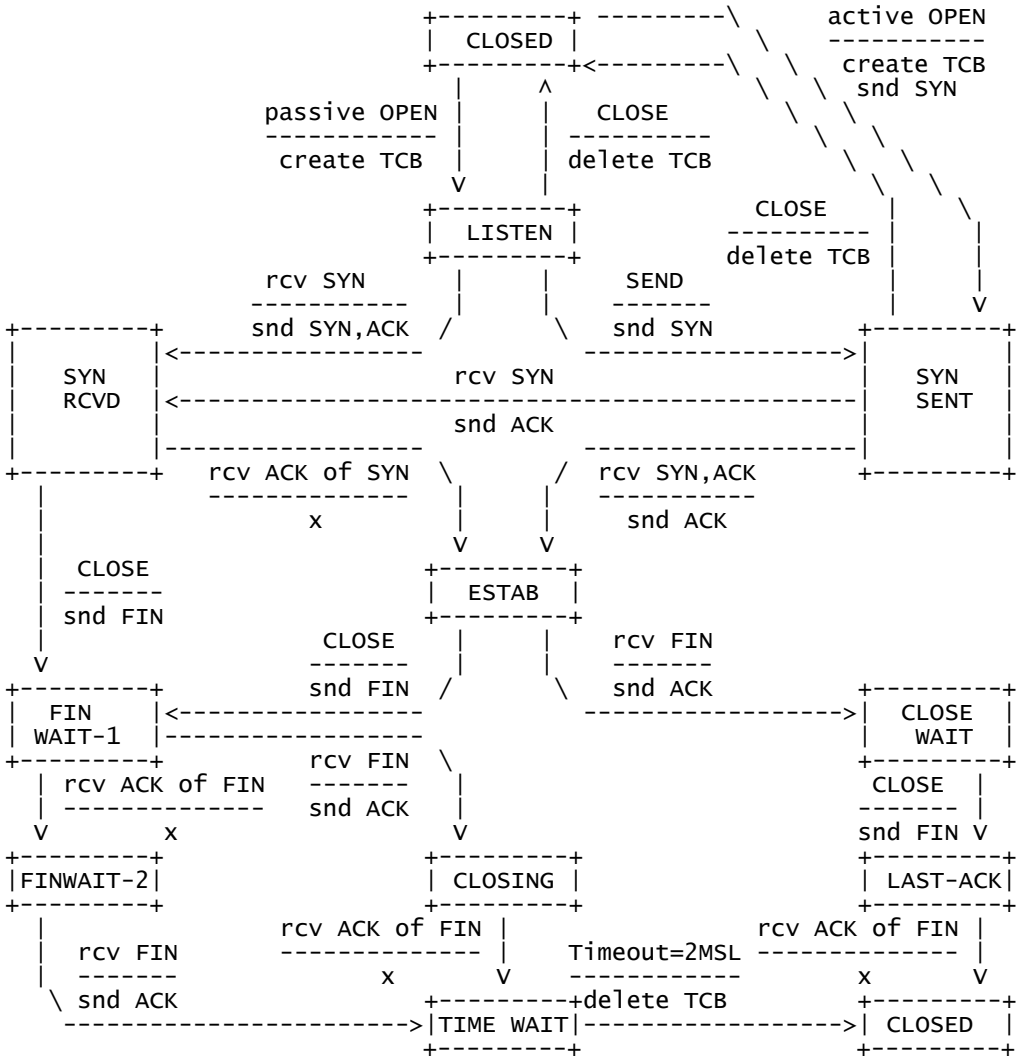
CLOSE-WAIT: Si somos nosotros los que **recibimos el paquete con el flag FIN levantado** pero aún **tenemos datos que enviar**, el socket entra en **estado CLOSE-WAIT**.

CLOSING: Si **ambos host desean finalizar la conexión** a la vez, los sockets entran en **estado CLOSING**.

LAST-ACK: Una vez enviados **sendos paquetes con el flag FIN levantado** en la finalización de una conexión TCP, cuando el último en haber enviado el paquete está **pendiente de recibir la confirmación**, entra en **estado LAST-ACK**.

TIME-WAIT: Una vez enviados **sendos paquetes con el flag FIN levantado** en la finalización de una conexión TCP, cuando el primero en haber enviado el paquete **envía la confirmación al último paquete FIN**, entra en **estado TIME-WAIT** para esperar un tiempo prudencial que le permita cerciorarse de la recepción del mismo.

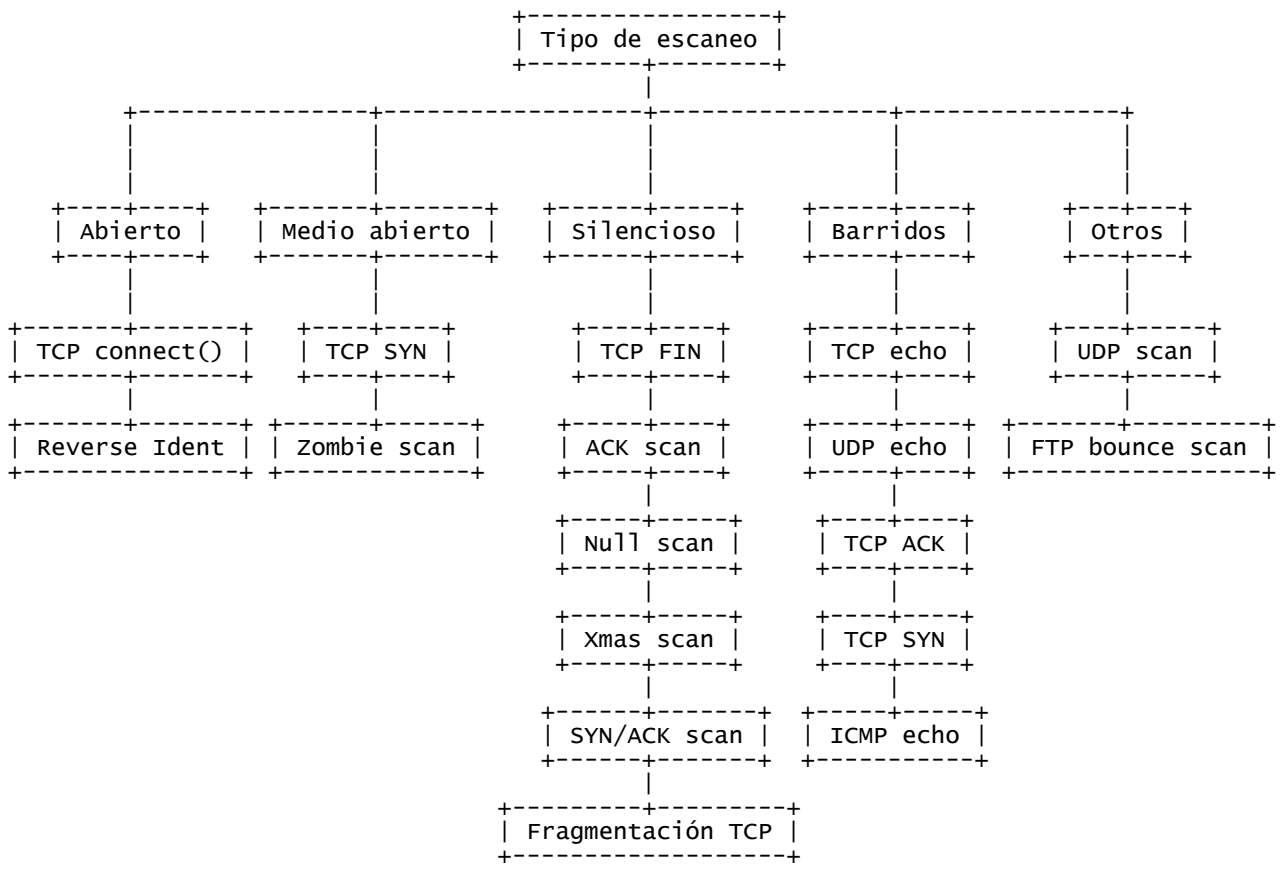
Para comprender mejor los estados TCP debemos echar un vistazo al diagrama de estados:


CLOSED
active OPEN
create TCB
snd SYN
passive OPEN
create TCB
CLOSE
delete TCB
LISTEN
CLOSE
delete TCB
rcv SYN
snd SYN,ACK
SEND
snd SYN
SYN RCVD
rcv SYN
snd ACK
SYN SENT
rcv ACK of SYN
x
rcv SYN,ACK
snd ACK
CLOSE
snd FIN
ESTAB
CLOSE
snd FIN
rcv FIN
snd ACK
FIN WAIT-1
CLOSE WAIT
rcv FIN
snd ACK
rcv ACK of FIN
x
CLOSE
snd FIN
FINWAIT-2
CLOSING
LAST-ACK
rcv ACK of FIN
x
rcv FIN
snd ACK
Timeout=2MSL
delete TCB
rcv ACK of FIN
x
TIME WAIT
CLOSED

Técnicas de escaneo

Ya conocemos los rudimentos técnicos en los que se basa cualquier escaneo de puertos. Es el momento de pasar a la parte divertida ;-). Vamos a echar un vistazo a las distintas técnicas de escaneo que existen, a su base técnica, a cómo realizarlas, así como a valorar los pros y los contras que conllevan. Comprenderemos que escaneando un mismo host de distintas formas obtenemos resultados distintos, así como porqué para realizar algunas de estas técnicas necesitamos unos privilegios especiales en el sistema.

Aquí tenemos un diagrama con la clasificación de las distintas técnicas de escaneo:



TCP connect()

Esta técnica es quizá la más común en cualquier software de escaneo de puertos. La técnica consiste en **usar la llamada *connect()* de TCP** para **intentar establecer una conexión con cada uno de los puertos** del host a escanear. Si **la conexión se establece**, el puerto está **abierto** (escuchando conexiones); en caso de recibir un aviso de **cierre de conexión** (RST), el puerto estará **cerrado**; y en caso de **no recibir respuesta**, se deduce que el puerto está **silencioso**.

Este tipo de escaneo es **extremadamente rápido**, pues puede realizarse de forma paralela para distintos puertos mediante el uso de **varios sockets**. Además, es un escaneo **fácil de implementar**.

Su principal desventaja es que es **llamativo en exceso**, pues resulta a todas luces llamativo establecer cientos o miles de conexiones en un margen de pocos segundos. Además, al realizarse intentos completos de conexión, cualquier sistema guardará **registros**.

Comportamiento del escaneo:

```
    host local ---[SYN]---> [O] Puerto TCP abierto en el host remoto
host local <---[SYN/ACK]--- [O] Puerto TCP abierto en el host remoto
    host local ---[ACK]---> [O] Puerto TCP abierto en el host remoto

  host local ---[SYN]---> [X] Puerto TCP cerrado en el host remoto
  host local <---[RST]--- [X] Puerto TCP cerrado en el host remoto

 host local ---[SYN]---> [~] Puerto TCP silencioso en el host remoto
                          -SIN RESPUESTA-
```

En nmap podemos invocar un escaneo **TCP connect()** mediante el comando:

nmap -vv -P0 -sT xxx.xxx.xxx.xxx

TCP SYN

Esta técnica, también conocida como **Half-open scan** (es el escaneo medio abierto por excelencia), es parecida a la anterior con la importante salvedad de no establecer completamente las conexiones.

En primer lugar, **se envía un paquete SYN** que finge intentar establecer una conexión y se espera la respuesta. **Si llega un paquete SYN/ACK** significa que el puerto está **abierto**; si llega un **paquete RST**, el puerto está **cerrado**; y si **no se recibe respuesta** se asume que está **silencioso**. En el caso de que el puerto esté abierto y recibamos el paquete SYN/ACK (es decir, están completos dos de los tres pasos del saludo en tres tiempos), nosotros **no responderemos con un paquete ACK** como sería lo esperado, sino que **mandaremos un paquete RST**. ¿Para qué? Pues precisamente para **evitar que se complete el inicio de conexión** y, por tanto, evitar que el sistema registre el suceso como un intento de conexión. En sistemas sin protección específica de cortafuegos o IDS, este escaneo suele pasar desapercibido.

Una característica importante de este escaneo es que **requiere elevados privilegios** en el sistema para poder lanzarlo, debido a que este tipo de paquetes usan **sockets TCP raw**. Por tanto, solo el ***root*** puede lanzar escaneos TCP SYN.

La principal ventaja de este tipo de escaneo es que suele ser **bastante discreto** y ofrece unos **resultados bastante buenos**.

Entre sus desventajas encontramos el que es algo **lento de realizar**, y que un sistema con un **firewall** o un **IDS** (aunque algunos muy básicos no) lo **detectará e identificará como escaneo** de puertos sin ninguna duda.

Comportamiento del escaneo:

```
    host local ---[SYN]---> [O] Puerto TCP abierto en el host remoto
host local <---[SYN/ACK]--- [O] Puerto TCP abierto en el host remoto
    host local ---[RST]---> [O] Puerto TCP abierto en el host remoto

  host local ---[SYN]---> [X] Puerto TCP cerrado en el host remoto
  host local <---[RST]--- [X] Puerto TCP cerrado en el host remoto

 host local ---[SYN]---> [~] Puerto TCP silencioso en el host remoto
                          -SIN RESPUESTA-
```

En nmap podemos invocar un escaneo **TCP SYN** mediante el comando:

nmap -vv -P0 -sS xxx.xxx.xxx.xxx

TCP FIN

El escaneo **TCP FIN**, también conocido como **Stealth scan** (se trata del escaneo silencioso más conocido), es uno de los más **discretos** que podemos encontrar dentro de las técnicas convencionales. Se apoya en una particularidad de los estándares internacionales de TCP/IP.

A la hora de realizar el escaneo, se **envía un paquete FIN** al puerto del host destino que queremos escanear. Los estándares de TCP/IP dicen que al recibir un paquete FIN en un puerto cerrado, se ha de responder con un paquete RST. Así pues, **si recibimos RST** por respuesta, el puerto está **cerrado**, y en caso de **no recibir respuesta** (se ignora el paquete FIN) el puerto **puede encontrarse abierto o silencioso**.

Ésto supone uno de los principales inconvenientes del escaneo TCP FIN, y es que **los puertos que nos figuran como abiertos, pueden estar en realidad en estado silencioso** (puesto que un puerto silencioso por definición ignora cualquier paquete recibido). Así pues, este tipo de escaneos **no obtienen unos resultados fiables**, y ese es su talón de Aquiles.

Otra gran desventaja de este sistema de escaneo viene de cierta compañía de software que tiene por costumbre pasarse por el forro cualquier estándar informático... sí, esa misma que estáis pensando: Microsoft. En los **sistemas Windows**, un puerto cerrado ignora los paquetes FIN, por lo que **escanear un sistema de este tipo con SYN FIN nos generará una enorme lista de puertos abiertos**, aunque realmente estén cerrados o silenciosos. Así que cuidado a la hora de usar esta técnica.

Como ventaja, tenemos el que estos escaneos **pasan desapercibidos en la gran mayoría de los firewalls**, al no intentar establecer ninguna conexión. Un IDS bien configurado, lo detectará.

Comportamiento del escaneo:

```
host local ---[FIN]---> [O] Puerto TCP abierto en el host remoto
                            -SIN RESPUESTA-

host local ---[FIN]---> [X] Puerto TCP cerrado en el host remoto
host local <---[RST]--- [X] Puerto TCP cerrado en el host remoto

host local ---[FIN]---> [~] Puerto TCP silencioso en el host remoto
                            -SIN RESPUESTA-
```

En nmap podemos invocar un escaneo **TCP FIN** mediante el comando:

nmap -vv -P0 -sF xxx.xxx.xxx.xxx

UDP scan

Esta técnica, frente a las demás técnicas orientadas a TCP, está **orientada al protocolo UDP** y sus puertos. Aunque a priori parezca que los puertos UDP no son muy interesantes, servicios como el rpcbind de Solaris, TFTP, SNMP, NFS... usan todos ellos UDP como protocolo de transferencia.

El sistema de escaneo consiste en **mandar un paquete UDP vacío** (0 bytes de datos) al puerto que deseamos escanear. Si el puerto está **cerrado**, el sistema **responderá con un paquete ICMP de tipo 3** (destino inalcanzable). En **caso de no responder**, el puerto puede estar **abierto** o **silencioso**.

Este sistema puede presentar un **grave problema de carencia de velocidad según en qué sistemas**, y es que en el **RFC #1812-"Requirements for IP version 4 routers"** (ftp://ftp.rfc-editor.org/in-notes/rfc1812.txt) se recomienda **limitar la capacidad de generación de mensajes ICMP de error**. En sistemas **Linux** (consultar el fichero **/ipv4/icmp.h** de las fuentes del kernel) esta limitación está fijada en unos **20 mensajes** por segundo. Sistemas como **Solaris** son más estrictos y tiene la limitación fijada en **2 por segundo**. Pero hay un sistema que, para variar, no hace mucho caso a los estándares, por lo que no tiene ninguna limitación prefijada... sí: Windows. Un escaneo UDP a un **sistema Windows** resulta **extremadamente rápido** como consecuencia de ello.

Comportamiento del escaneo:

```
host local ---{UDP}---> {O} Puerto UDP abierto en el host remoto
                          -SIN RESPUESTA-

   host local ---{UDP}---> {X} Puerto UDP cerrado en el host remoto
host local <---|ICMP #3|--- {X} Puerto UDP cerrado en el host remoto

host local ---{UDP}---> {~} Puerto UDP silencioso en el host remoto
                          -SIN RESPUESTA-
```

En nmap podemos invocar un escaneo **UDP** mediante el comando:

nmap -vv -P0 -sU xxx.xxx.xxx.xxx

ACK scan

La mayoría de las técnicas de escaneo nos permiten identificar con exactitud los puertos abiertos o cerrados, pero generalmente los **puertos silenciosos** no se pueden identificar con claridad. El **escaneo ACK** está destinado a **identificar de forma precisa cuándo un puerto se encuentra en estado silencioso**. Esta técnica es usada también para poder escanear hosts que estén detrás de un firewall que bloquee los intentos de conexión (paquetes SYN).

Su funcionamiento se basa en el **envío de paquetes ACK** con **números de secuencia y confirmación aleatorios**. Cuando reciba el paquete, si el puerto se encuentra **abierto**, **responderá con un paquete RST**, pues no identificará la conexión como suya; si el puerto está **cerrado responderá con un paquete RST**, pero **si no se obtiene respuesta** (obviamente primero debemos asegurarnos que el host está en línea) podemos identificar claramente el puerto como **filtrado** (**puerto silencioso**).

Normalmente el escaneo ACK se realiza como **apoyo a un escaneo anterior**, para determinar los puertos silenciosos y poder **identificar** mediante una combinación de técnicas **el estado real de todos ellos**. Por ejemplo, ante un host con un firewall que bloquee intentos de conexión (SYN), podemos realizar un FIN scan para determinar los puertos cerrados, y después un ACK scan para determinar qué puertos están abiertos y cuáles silenciosos.

Esta técnica también es usada como **variante del ping** (ICMP echo) de toda la vida, para saber **si un host está activo** (recibiremos **respuesta RST**) o **no** (cuando **no hay respuesta** o la respuesta es **destino inalcanzable**).

Comportamiento del escaneo:

```
host local ---[ACK]---> [O] Puerto TCP abierto en el host remoto
host local <---[RST]--- [O] Puerto TCP abierto en el host remoto

host local ---[ACK]---> [X] Puerto TCP cerrado en el host remoto
host local <---[RST]--- [X] Puerto TCP cerrado en el host remoto

host local ---[ACK]---> [~] Puerto TCP silencioso en el host remoto
                          -SIN RESPUESTA-
```

En nmap podemos invocar un escaneo **ACK** mediante el comando:

nmap -vv -PT xxx.xxx.xxx.xxx

Null scan

Este escaneo tiene muchos **puntos en común con el escaneo FIN**. Su funcionamiento base es el mismo: **enviamos un paquete malformado** (en este caso se trata de un paquete TCP con **todos los flags desactivados**) y esperamos la respuesta. En caso de que el puerto destino esté **cerrado**, nos **responderá con un paquete RST**; y en caso de **no recibir nada** (nuestro paquete es ignorado), se trata de un puerto **abierto o silencioso**.

La ventaja frente al escaneo FIN radica en que ciertos firewalls vigilan los paquetes de finalización de conexión además de los de establecimiento, de forma que el escaneo nulo podrá realizarse allí dónde el FIN no sería posible. El resto de ventajas y desventajas son las mismas que en el escaneo FIN.

Comportamiento del escaneo:

```
host local ---[   ]---> [O] Puerto TCP abierto en el host remoto
                            -SIN RESPUESTA-

host local ---[   ]---> [X] Puerto TCP cerrado en el host remoto
host local <---[RST]--- [X] Puerto TCP cerrado en el host remoto

host local ---[   ]---> [~] Puerto TCP silencioso en el host remoto
                            -SIN RESPUESTA-
```

En nmap podemos invocar un escaneo **Null** mediante el comando:

nmap -vv -P0 -sN xxx.xxx.xxx.xxx

Xmas scan

El escaneo Xmas se basa también en el principio de la respuesta RST por parte de un puerto cerrado al recibir un paquete incorrecto (como el escaneo FIN). En el caso del **escaneo Xmas**, se trata de un **paquete con los flags FIN, URG y PSH activados** (aunque ciertas implementaciones activan FIN, URG, PSH, ACK y SYN e incluso algunas activan todos los flags). Podría decirse que es lo contrario del escaneo Null, pero logrando el mismo efecto.

Al igual que el escaneo Null, se usa bajo ciertas circunstancias en las que el escaneo FIN no es posible; y también comparte con éstos sus particularidades.

Comportamiento del escaneo:

```
host local ---[xmas]---> [O] Puerto TCP abierto en el host remoto
                            -SIN RESPUESTA-

host local ---[xmas]---> [X] Puerto TCP cerrado en el host remoto
 host local <---[RST]--- [X] Puerto TCP cerrado en el host remoto

host local ---[xmas]---> [~] Puerto TCP silencioso en el host remoto
                            -SIN RESPUESTA-
```

En nmap podemos invocar un escaneo **Xmas** mediante el comando:

nmap -vv -P0 -sX xxx.xxx.xxx.xxx

SYN/ACK scan

Este tipo de escaneo tiene una **base parecida** a los anteriormente citados **FIN**, **Null** y **Xmas**, pero con la sustancial diferencia de que en este caso **los paquetes malformados fingen ser un error** en la transacción de una conexión legítima. Mediante esta técnica, **se envía un paquete SYN/ACK** al puerto que deseamos escanear en el host remoto. Si el puerto se encuentra **cerrado**, nos **responderá con un paquete RST**. En caso de estar **abierto** o **silencioso**, simplemente **ignorará el paquete** y no obtendremos respuesta.

Como ventaja, este tipo de escaneo **evade la mayoría de firewalls e IDS sencillos**, pero comparte con los escaneos anteriormente citados sus problemas, principalmente la **falta de fiabilidad** a la hora de determinar los **puertos abiertos o silenciosos**.

Comportamiento del escaneo:

```
host local ---[SYN/ACK]---> [O] Puerto TCP abierto en el host remoto
                           -SIN RESPUESTA-

host local ---[SYN/ACK]---> [X] Puerto TCP cerrado en el host remoto
    host local <---[RST]--- [X] Puerto TCP cerrado en el host remoto

host local ---[SYN/ACK]---> [~] Puerto TCP silencioso en el host remoto
                           -SIN RESPUESTA-
```

En nmap **no se encuentra implementado** este tipo de escaneo.

Ping sweep

Un **ping sweep** (también llamado **barrido de ping**) **no** es en realidad una técnica de escaneo de puertos... sino más bien una técnica de **escaneo de hosts**.

Es el momento de hablar de una opción de **nmap** que he incluido en el ejemplo de todos los escaneos de los que hemos hablado hasta ahora. Es la opción ***-P0***. Veamos qué dice la guía de referencia rápida de **nmap** (***nmap -h***) al respecto de este comando...

```
-P0 Don't ping hosts (needed to scan www.microsoft.com and others)
```

Efectivamente, mediante esta opción logramos que, antes de realizar el escaneo de puertos propiamente dicho, el software **no compruebe si el host está activo**. Y os preguntaréis, ¿para qué demonios me gustaría a mí que no se realizara esa comprobación? Pues es una técnica muy común el denegar, vía firewall, la **salida de paquetes ICMP echo reply**. Así, al realizar un ping a un host, éste no responde y puede parecer que esté inactivo. Probad a realizar un ping a www.microsoft.com y luego visitad su web.

Ahora imaginad que **el objetivo** de nuestro escaneo va a ser **toda una red** (por ejemplo 192.168.0.0/24), en lugar de un único host. Lo primero que nos interesará es saber **qué hosts están activos**, pues **si escaneamos toda la red** con la opción ***-P0***, perderemos mucho tiempo mandando paquetes y esperando la respuesta de equipos que en realidad están inactivos. Pero cabe la posibilidad de que algunos equipos nos hagan creer que están inactivos cuando en realidad no lo están... y aquí es donde entran las diversas técnicas de **ping sweep**.

Mediante esta técnica, se realiza un **barrido** comprobando qué host dentro de un rango se encuentran **activados**. Los métodos para comprobar esto son varios:

TCP echo: Envío de paquetes **TCP** al puerto echo (**TCP/7**). Si **recibe respuesta**, el host está activo.
UDP echo: Envío de paquetes **UDP** al puerto echo (**UDP/7**). Si **recibe respuesta**, el host está activo.
TCP ACK: Envío de paquetes **TCP ACK**. Si se obtiene **respuesta RST**, el host está activo.
TCP SYN: Envío de paquetes **TCP SYN**. Si se obtiene **respuesta RST o SYN/ACK**, el host está activo.
ICMP echo: Este es el ping de toda la vida. **ICMP echo request** e **ICMP echo reply**.

Aunque todas ellas son válidas, incluso el ping clásico, **las más efectivas son TCP ACK y TCP SYN**. **TCP echo y UDP echo no son muy usadas** ni útiles, pues prácticamente ningún host tendrá abierto el puerto echo y si lo tiene, es poco probable que bloquee los intentos de ping normales.

En nmap podemos invocar un **ping sweep** mediante estos comandos:

TCP ACK: ***nmap -vv -sP -PT xxx.xxx.xxx.xxx/xx***
TCP SYN: ***nmap -vv -sP -PS xxx.xxx.xxx.xxx/xx***
ICMP echo: ***nmap -vv -sP -PI xxx.xxx.xxx.xxx/xx***
TCP ACK e ICMP echo en paralelo: ***nmap -vv -sP -PB xxx.xxx.xxx.xxx/xx***

La técnica **TCP ACK e ICMP echo en paralelo** realiza a la vez ambas técnicas, de forma que se pueda **evadir un firewall** que implemente protección contra **una de esas técnicas**. Este modo es el usado por **nmap** en caso de no especificar ninguno (***nmap -vv -sP xxx.xxx.xxx.xxx/xx***).

Es útil **reforzar la exploración** del modo -PB con un **ping TCP SYN** (-PS), pues ciertos firewalls bloquean tanto los intentos de ping ICMP echo como TCP ACK.

Técnicas avanzadas

Ya conocemos, además de las bases técnicas, una gran variedad de técnicas "estándar" de escaneo de puertos: las más sencillas y comunes. Pero existen otras **técnicas más complejas** que ahora vamos a tratar. El que sean más complejas **no es sinónimo de mayor efectividad**, pues ya hemos visto varias veces que no hay un único escaneo que sea útil para todo, sino que la técnica a usar depende de la situación... y casi siempre lo mejor es **usar una combinación de técnicas**. Echemos un vistazo a estas técnicas "especiales":

Reverse Ident

Antes de hablar del escaneo Ident inverso, debemos hablar del **Protocolo de Identificación** que está definido en el **RFC #1413 -"Identification Protocol"** (ftp://ftp.rfc-editor.org/in-notes/rfc1413.txt). El fin del protocolo Ident es proporcionar **información acerca de la identidad del usuario de una conexión TCP**, para lo cual existe un demonio a la escucha al que, enviando una **query** en una determinada estructura (definida en el RFC), devuelve la información del usuario. Esto es lo que se llama Ident... ¿y entonces qué es Ident inverso?

Mediante **Ident inverso** somos nosotros los que **establecemos una conexión con el host remoto** y luego preguntamos a Ident por su usuario. De cara al host remoto, el usuario de esa conexión seguirá siendo el usuario de su sistema, aunque la conexión la hayamos establecido nosotros. Dado que esta técnica **requiere que se establezca completamente una conexión TCP**, su base es el escaneo **TCP connect()**. Una vez establecida la conexión con el puerto en el host remoto, redirigimos una query al puerto Ident y obtenemos así información bastante interesante sobre quién es el usuario tras esa conexión. Obviamente no tiene el mismo interés un demonio corriendo con privilegios de nobody (httpd) o con privilegios de root...

Es importante tener en cuenta que **no todos los hosts corren el servicio de Ident**, y de los que lo hacen, muchos usan algún tipo de sistema de identificación. No obstante, puede resultar muy útil bajo determinadas circunstancias...

Ésta técnica fue descrita por primera vez por Dave Goldsmith en 1996, en un correo a la lista de **Bugtraq**.

Comportamiento del escaneo:

```
    host local ---[SYN]---> [O] Puerto TCP abierto en el host remoto
host local <---[SYN/ACK]--- [O] Puerto TCP abierto en el host remoto
    host local ---[ACK]---> [O] Puerto TCP abierto en el host remoto
  host local ---[query]---> [O] Puerto TCP/113 abierto en el host remoto
  host local <---[Ident]--- [O] Puerto TCP/113 abierto en el host remoto

  host local ---[SYN]---> [X] Puerto TCP cerrado en el host remoto
  host local <---[RST]--- [X] Puerto TCP cerrado en el host remoto

 host local ---[SYN]---> [~] Puerto TCP silencioso en el host remoto
                          -SIN RESPUESTA-
```

En nmap podemos invocar un escaneo **reverse Ident** mediante el comando:

nmap -vv -P0 -sT -I xxx.xxx.xxx.xxx

Zombie scan

Este tipo de escaneo, también conocido como **Dumb scan**, **IP ID Header scan**, e **Idle scan** fue descrito por primera vez por **antirez** en un correo a la lista de **Bugtraq**. Su funcionamiento es bastante **ingenioso** (y algo rebuscado) y basa su técnica en **particularidades de la pila TCP/IP** de la mayoría de los sistemas operativos. Normalmente el escaneo dumb se realiza **basado en la técnica del escaneo SYN**, pero nada nos impediría realizarlo con cualquier otra.

Requisito indispensable para poder llevar a cabo este escaneo es poder contar con un **host zombie** (**dummy host**, **dumb host**...). ¿Y qué es un host zombie? Al contrario que un host bastión, un host zombie es aquel que estando online **tiene un tráfico muy bajo** o (mejor aún) nulo. Hay que decir que encontrar un host de este tipo es muy complicado y requiere buenas dosis de paciencia y de **ping sweep**...

Por tanto, en este escaneo participan tres hosts: **hostA** (nosotros), **hostB** (host zombie) y **hostC** (host remoto a escanear).

En primer lugar, nos aseguramos de que **hostB es un host zombie realmente**, y para ello le lanzamos un **ping** que nos permita analizar el **campo ID encapsulado en la cabecera IP**:

```
60 bytes from BBB.BBB.BBB.BBB: seq=1 ttl=64 id=+1 win=0 time=96 ms
60 bytes from BBB.BBB.BBB.BBB: seq=2 ttl=64 id=+1 win=0 time=88 ms
60 bytes from BBB.BBB.BBB.BBB: seq=3 ttl=64 id=+1 win=0 time=92 ms
```

Vemos que el **incremento de la ID** es de uno en cada paso del ping. Podemos **asumir que el hostB no tiene tráfico**. Ahora es cuando viene lo interesante del escaneo: **mandamos a hostC un paquete SYN falseado** (packet spoofing) con dirección **origen hostB**. El comportamiento de **hostC** será parecido al caso de un escaneo SYN estándar:

```
hostB <---[SYN/ACK]--- [O] Puerto TCP abierto en hostC

hostB <---[RST/ACK]--- [X] Puerto TCP cerrado en hostC
```

Así pues, **hostB** se encontrará con un paquete que **no esperaba** (él no ha lanzado ningún intento de conexión). La **reacción de hostB** viene dada por su implementación de la pila TCP/IP y depende del paquete recibido:

```
hostB <---[SYN/ACK]--- [O] Puerto TCP abierto en hostC
    hostB ---[RST]---> [O] Puerto TCP abierto en hostC

hostB <---[RST/ACK]--- [X] Puerto TCP cerrado en hostC
                  -SIN RESPUESTA-
```

Así pues, si el puerto escaneado en **hostC** está **abierto**, estaremos **forzando a hostB a enviar un paquete** de respuesta, mientras que si el puerto de **hostC** está **cerrado**, **hostB ignorará el paquete RST/ACK** y no enviará nada. Obviamente en el caso de que el puerto de **hostC** esté **silencioso** no habrá **ningún tipo de envío** de tráfico. ¿Cómo podemos saber qué flags fueron enviados entre hostC y hostB desde hostA? Pues lo sabremos porque durante todo este proceso, estaremos manteniendo un ping en paralelo constante con hostB.

Si el **puerto de hostC está abierto** (hostC manda SYN/ACK a hostB y éste responde con RST):

```
60 bytes from BBB.BBB.BBB.BBB: seq=25 ttl=64 id=+1 win=0 time=92 ms
60 bytes from BBB.BBB.BBB.BBB: seq=26 ttl=64 id=+3 win=0 time=80 ms
60 bytes from BBB.BBB.BBB.BBB: seq=27 ttl=64 id=+2 win=0 time=83 ms
```

Si el **puerto de hostC está cerrado** (hostC manda RST/ACK y hostB no responde) ó el puerto de hostC está **silencioso** (no manda nada):

```
60 bytes from BBB.BBB.BBB.BBB: seq=25 ttl=64 id=+1 win=0 time=92 ms
60 bytes from BBB.BBB.BBB.BBB: seq=26 ttl=64 id=+1 win=0 time=80 ms
60 bytes from BBB.BBB.BBB.BBB: seq=27 ttl=64 id=+1 win=0 time=83 ms
```

Observando el **campo ID** vemos que, en caso de que el **puerto de hostC esté abierto** y obligar a hostB a mandar un paquete, **el incremento es mayor** que en el caso de que no envíe nada. Podemos asumir por tanto, que en el caso de que el incremento de ID sea mayor en un determinado momento, nos indica que el puerto escaneado estaba abierto.

Este complejo método de escaneo es muy ingenioso, porque de cara a hostC, el escaneo provino de hostB, y de cara a hostB, nosotros solamente estábamos haciendo ping...]:-)

En nmap podemos invocar un escaneo **zombie** mediante el comando:

nmap -vv -P0 -p- -sI zom.zom.zom.zom xxx.xxx.xxx.xxx

FTP bounce scan

El escaneo FTP bounce se parece en cierto modo al escaneo zombie... pero en su forma son totalmente distintos. El escaneo FTP bounce se basa en unas peculiaridades del protocolo FTP, descrito en el **RFC #959 -"File Transfer Protocol"** (ftp://ftp.rfc-editor.org/in-notes/rfc959.txt).

Cuando establecemos una **conexión FTP**, en primer lugar se realiza una conexión entre el cliente y el **puerto de control de datos del servidor** (normalmente TCP/21). Pero para **poder realizar transferencias** de ficheros entre cliente y servidor, es necesario **establecer otra conexión de datos**. Esto, mediante comandos RAW, se realiza con el **comando PORT**, que indica la IP y el puerto con el que el servidor puede establecer la conexión de datos. El "fallo" está en que **mediante el comando PORT, podemos indicar una IP cualquiera**, aunque no sea desde la que se inició la conexión.

Las consecuencias de ello son que **podemos establecer una conexión de control de datos** con un servidor FTP (mejor si es anónimo) y, mediante el comando **PORT**, **intentar establecer conexiones de datos** con los distintos puertos de la máquina a escanear. **Según la respuesta** que nos devuelva el servidor FTP, el puerto se encontrará abierto o cerrado. En caso de que el puerto esté **abierto**, el servidor FTP **podrá establecer una conexión de datos** y responderá "**226 Transfer complete.**"; y en caso de que el puerto esté **cerrado** y el servidor **no pueda establecer la conexión**, responderá "**425 Can't open data connection.**".

Esta técnica fue descrita por primera vez por ***Hobbit*** en 1985

Comportamiento del escaneo:

```
    host local
(PORT apuntando a host remoto)
     | | | | |
     ! ! ! ! !
   Servidor FTP <---------> [O] Puerto TCP abierto en el host remoto
     | | | | |
     ! ! ! ! !
 (226 Transfer complete.)
    host local


    host local
(PORT apuntando a host remoto)
     | | | | |
     ! ! ! ! !
   Servidor FTP <---------> [X] Puerto TCP cerrado en el host remoto
     | | | | |
     ! ! ! ! !
(425 Can't open data connection.)
    host local
```

En nmap podemos invocar un escaneo **FTP bounce** mediante el comando:

nmap -vv -P0 -b usuario@ftp.ftp.ftp.ftp:puerto xxx.xxx.xxx.xxx

Fragmentación TCP

La **fragmentación TCP no es un método de escaneo en sí mismo**, sino una técnica avanzada de **ocultación del escaneo**. Se puede usar con **cualquier tipo de escaneo** TCP, aunque normalmente se utiliza con los escaneos SYN, FIN, Null y Xmas.

La técnica en sí misma consiste en **fragmentar** (dividir) **la propia cabecera TCP** en fragmentos más pequeños. Generalmente se envía en primera instancia un paquete de 64 bits con el puerto de origen y destino, para enviar después la inicialización de las banderas TCP. El **reensamblado** del paquete se realiza mediante **IPM (Internet Protocol Module)** analizando equivalencias entre campos de la cabecera TCP (origen, destino, protocolo e identificación).

Su gran ventaja reside en que la **mayoría de los IDS** detectan los escaneos de puertos mediante **sistemas de firmas**, por lo que dividiendo las cabeceras TCP lograremos evadir estos sistemas. Pero esta técnica también conlleva **grandes problemas** y resultados extraños e imprevisibles, pues **ciertos hosts** pueden ser **incapaces de parsear y reensamblar** los fragmentos que le enviamos, y se pueden producir **cuelgues**, **reinicios**, e incluso **volcados de monitorización de los dispositivos de red**. Lo cual bajo ciertas circunstancias (potenciales ataques) puede ser bueno, pero para realizar un escaneo no lo es. Además, **los paquetes TCP fragmentados pueden ser bloqueados por colas de fragmentación en el kernel** o detenidos por ciertos tipos de firewalls correctamente configurados.

En nmap podemos invocar un escaneo de **fragmentación TCP** (en este caso SYN) mediante el comando:

nmap -vv -P0 -sS -f xxx.xxx.xxx.xxx

Detección del fingerprint

Siempre ha resultado interesante el **saber el sistema operativo** (y su versión) y **la arquitectura** que corre un host a la hora de realizar una auditoría del sistema. Un bug en un demonio puede ayudarnos a obtener una shell de root en el host, pero **la shellcode no depende del demonio sino del sistema operativo**...

El **método clásico** para averiguar qué sistema operativo corre un determinado host era el de **consultar los banner** de las conexiones. Al conectar a un servidor de telnet, servidor web (*GET / HTTP/1.0\n*), servidor FTP... era (y es) muy habitual el encontrarse con una **línea de información (banner)** que nos diga **la versión del software y la del sistema operativo**. Esta técnica **no resulta fiable** en absoluto pues para empezar estos banner cada vez están más **limitados**, y además es una información **fácilmente falsificable** (es divertido tener un Apache que diga que es un IIS vulnerable al code/decode... :-P). Pero hoy en día tenemos otra técnica, una **poderosa herramienta** para determinar el sistema operativo de un host: **el análisis de la huella digital (fingerprint) de su pila TCP/IP**.

Este análisis es posible gracias a las **particularidades de los distintos sistemas** a la hora de manejar la **pila de protocolos TCP/IP**. A lo largo del manual he comentado en varias ocasiones lo aficionados que son en Redmond a pasarse por el forro los estándares... bien, pues eso nos va a venir de perlas a la hora de identificar una máquina Windows. ;-)

Veamos las distintas técnicas de análisis de fingerprint:

La prueba FIN

Se manda un paquete con el **flag FIN** a un puerto abierto. El estándar definido por el **RFC de TCP** es **no responder**, pero ciertos sistemas que no cumplen el estándar (Microsoft Windows, BSDI, CISCO, HP/UX, MVS, e IRIX) responden con un paquete **RST**.

La prueba de la bandera BOGUS

Se manda un paquete **malformado** con un **flag TCP sin definir** (**BOGUS flag**) en la cabecera (generalmente el **bit 7**, antes se usaba el 8 pero ahora es usado como campo ECN) de un **paquete SYN**. Linux anterior a 2.0.35 responden manteniendo la bandera BOGUS, y ciertos sistemas resetean la conexión. El resto, eliminan la bandera malformada.

La prueba del número de secuencia inicial

Se analiza el **número de secuencia inicial de una conexión TCP**. Las posibles respuestas son **64k** (UNIX antiguos), **incrementos aleatorios** (Solaris modernos, IRIX, FreeBSD, Digital UNIX, Cray...), **aleatorios** (Linux a partir de 2.0.*, OpenVMS, AIX modernos...), **modelos dependientes del tiempo** con incrementos regulares (Microsoft Windows...) e incluso **números de secuencia iniciales constantes** (ciertos hubs 3com, impresoras Apple LaserWriter...).

La prueba IP ID

Ya hablamos del **IP ID** en el **escaneo zombie**, así que ya sabemos qué es. El IP ID **puede calcularse** mediante un **incremento de sistema por cada paquete** mandado, ser **aleatorio** (OpenBSD), ser **0** en caso de marcar el **bit de no fragmentación** (Linux), **incrementos fijos** por cada paquete (Microsoft Windows) e incluso **aleatorios**. El que el IP ID sea fácilmente predecible es muy positivo a la hora de lanzar un escaneo zombie...

La prueba TCP timestamp

Se analiza la **opción TCP timestamp** en una conexión TCP. Ciertos sistemas **no la soportan**, otros realizan **incrementos fijos** (2hz, 100hz, 1000hz...) y otros devuelven siempre un valor de **0**.

La prueba del bit de no fragmentación

Algunos sistemas **activan el bit de no fragmentación** en ciertos paquetes de los que mandan. Analizando los casos en que esto ocurre se obtiene información del sistema.

La prueba del tamaño inicial de ventana

Observando el **tamaño inicial de ventana** en las conexiones TCP se puede obtener información del sistema. Por ejemplo AIX usa **0x3F25**, otros como Microsoft Windows, FreeBSD y OpenBSD usan **0x402E**...

La prueba del número de confirmación

Aunque **debería ser estándar**, también hay ciertos sistemas que se toman "licencias" a la hora de implementar esta característica. Enviando un paquete **FIN/PSH/URG** se debería obtener como **respuesta** un **número de confirmación** que coincida con el **número de secuencia** enviado... aunque ciertas impresoras y Microsoft Windows envían el **número de secuencia inicial incrementado en una unidad** (+1). Otro caso es el comportamiento de Microsoft Windows mandarle un paquete **SYN/FIN/URG/PSH**: unas veces devolverá el número de confirmación **correcto**, otras veces le **sumará una unidad**, otras veces responderá con un número de confirmación **aleatorio**...

La prueba del control de mensajes de error ICMP

Esta prueba se basa en lo ya comentado en el **escaneo UDP**. Examinando las **limitaciones de mensajes de error ICMP** podemos obtener información del sistema: Linux limita los mensajes a **80 cada 4 segundos**, Solaris los limita a **2 por segundo**, Microsoft Windows **no los limita**...

La prueba de referencia de mensajes ICMP

El estándar del RFC fija que los **mensajes de error ICMP** deben hacer una **pequeña referencia** a un mensaje ICMP que causa varios errores. Para mensajes de **puerto inalcanzable**, casi todos los sistemas mandan la **cabecera IP más 8 bytes**. Pero Solaris manda algunos bytes más, y Linux más aún, lo que permite reconocerlos aún sin ningún puerto abierto.

La prueba de eco de integridad de mensajes de error ICMP

Esta prueba se basa en la misma **referencia de mensaje** que la anterior. Ciertos sistemas usan la cabecera que les envías como **espacio para escribir** en el procesamiento inicial, lo que hace que se devuelvan un tanto modificados. AIX y BSDI modifican el campo de **"longitud total"**, algunos BSDI, FreeBSD, OpenBSD, ULTRIX, y VAXen estropean la **IP ID** enviada, y también FreeBSD y AIX mandan **checksums inconsistentes** o a 0 (al cambiar el TTL tampoco iban a resultar válidas...).

La prueba del tipo de servicio

En mensajes de **ICMP puerto inalcanzable**, el **tipo de servicio (TOS)** se establece a 0 de forma estándar. Sin embargo, **Linux usa 0xC0** que no es un estándar de valor TOS sino parte del campo de precedencia (que no es usado).

La prueba de manejo de fragmentación

Cuando un sistema recibe **paquetes fragmentados**, el **orden de preferencia a la hora de sobreescribir** los datos repetidos caracteriza también al sistema. Algunos dan preferencia a los datos **ya existentes** en buffer y otros dan preferencia a los **datos nuevos**.

La prueba de las opciones TCP

Las **opciones TCP** son una de las mejores formas para distinguir a los sistemas. El motivo viene dado porque **no todos los sistemas implementan todas** las opciones, o **no las implementan de igual forma**, o **no las tratan de igual forma**... además, al poder **incluirse todas en un solo paquete** (*Window Scale=10; NOP; Max Segment Size = 265; Timestamp; End of Ops;*) nos permite obtener datos muy interesantes de una forma muy rápida.

Sistemas como FreeBSD o Linux superior a 2.1.x soportan **todas**, mientras que Linux inferiores a 2.0.x no soportan **prácticamente ninguna**. Pero como cada vez es más común que los sistemas implementen todas las opciones, se puede también estudiar el valor de las mismas. Mandando un **MSS (Max Segment Size) muy pequeño**, los sistemas Linux **lo aceptarán y responderán el eco**, pero otros sistemas **lo rechazarán y responderán con valores más altos**. Aún obteniendo las mismas opciones y los mismos valores, puede que el orden resulte alterado: Solaris responde con **NNTNWME** (*<no op><no op><timestamp><no op><window scale><echoed MSS>*) mientras que Linux superior a 2.1.x responde con **MENNTNW**.

La prueba de cronología de exploits

A la hora de distinguir entre la **pila TCP/IP de sistemas Windows95, Windows98 y WindowsNT** existen bastantes problemas. A pesar del tiempo pasado entre versiones, la pila TCP/IP entre 95 y 98 **no cambió absolutamente nada**, y para NT **cambió muy poco**, con todo (lo malo) que ello implica.

Pero aún así es posible distinguir entre estos sistemas por lo vulnerables que han sido, en orden cronológico, a distintos ataques de **denegación de servicio**. Sí, es penoso, ¿verdad? xD. Lanzando ataques como **Ping Of Death**, **WinNuke**, **Teardrop** o **Land** y comprobando después si el sistema sigue en pie se puede tomar una idea de qué tipo de sistema Windows se trata.

La prueba de resistencia a SYN flood

Ciertos sistemas son más sensibles que otros al llamado **SYN flood (inundación SYN)**. Cuando mandamos **muchos paquetes SYN falsificados** (para evitar que nuestro propio sistema corte la conexión) a un mismo puerto en un host remoto, es común que éste se bloquee y **no permita** aceptar nuevas conexiones. Algunos sistemas como Linux solucionan esto mediante el uso de **SYN cookies**, por lo que no resultan vulnerables.

En nmap podemos invocar la **detección del fingerprint** mediante el comando:

nmap -vv -P0 -O xxx.xxx.xxx.xxx

Protección frente a escaneos

Ahora sabemos cómo **realizar escaneos** de puertos en toda clase de condiciones, cómo **interpretar los resultados** de distintas técnicas y cómo **combinarlas** para obtener resultados bastante fiables. Es hora de saber cómo **proteger nuestro propio sistema** de éstas mismas técnicas.

Lo primero que debemos decir es que **no es malo tener puertos abiertos**: sin puertos abiertos Internet no existiría pues nadie podría conectarse con otra persona. Pero hay que tener unas **normas básicas** a la hora de ofrecer servicios, abrir y cerrar puertos...

1- Abrir solamente los puertos necesarios: Los únicos puertos abiertos en nuestro sistema deben ser los necesarios para que funcionen correctamente los servicios que queremos ofrecer desde nuestra máquina. Si deseamos además evitar miradas indiscretas, mejor si los colocamos a la escucha en puertos no estándar.

2- Controlar los puertos abiertos dinámicamente: Cualquier aplicación que realiza conexiones debe abrir de forma dinámica puertos para poder realizar la conexión. Normalmente se utilizan puertos arbitrarios superiores al 1024, pero en ciertas circunstancias (por ejemplo para el DCC de IRC) conviene controlar el rango de puertos permitidos o asignarlos a mano (por ejemplo para aplicaciones P2P).

3- Proteger los puertos que no usemos: El resto de los puertos deberán estar cerrados, o mejor aún, silenciosos.

4- Proteger el acceso a los servicios que deban ser restringidos: Si debemos ofrecer un servicio pero no de forma pública, éste debe ser protegido mediante sistemas de autentificación adecuados.

5- Proteger las conexiones: Siempre que sea posible, usar conexiones cifradas: SSL es vuestro amigo, no le abandonéis: él nunca lo haría. :-)

6- Usar un firewall: Imprescindible su uso, bien se traten de máquinas independientes o dependientes del propio host. Nadie tiene excusa para no usar uno, pues existen muchos firewalls de escritorio gratuitos.

7- Usar un IDS: Un firewall protege nuestra red según las reglas con las que esté diseñado. Un IDS detecta (y actúa en consecuencia) ataques según una serie de reglas programadas por nostros. La combinación perfecta: firewall + IDS.

8- Usar esquemas de seguridad redundantes: ¿Qué mejor que un firewall? ¡Dos! Si tienes un router que actúa como firewall, instala otro en el sistema operativo de tu ordenador: dos barreras suponen mayores complicaciones que una.

9- Ocultar información sensible: Cuanta menos información tenga un potencial atacante de nuestro sistema, mejor para nosotros. Hay que desactivar los banners de información de cualquier servicio, ocultar las versiones... incluso se puede, mediante IPtables, falsificar la huella de la pila TCP/IP para engañar a los sistemas de detección de fingerprint que vimos en el punto anterior. Es un tema interesantísimo... quizá me ponga a ello para otro artículo... :-m~

10- Usar sistemas de seguridad avanzados: Aprovechar los últimos sistemas de seguridad. Imaginad que queremos ofrecer un servicio pero no tener absolutamente ningún puerto abierto. ¿Imposible? No. Existe una técnica llamada port knocking mediante la cual un sistema con todos los puertos cerrados tiene un demonio a la escucha de los logs del firewall. Cuando éste reconoce una secuencia determinada de intentos de conexión a determinados puertos y en un determinado orden, abre un puerto, establece un socket para abrir una conexión con el host remoto y vuelve a cerrarlo para que nadie más pueda conectarse al servicio. Otro tema muy interesante...

***- Tener el software actualizado:** De nada le sirve a un atacante saber la versión de nuestro Apache si no hay fallos conocidos. Conviene así mismo estar al tanto de estos fallos para reaccionar un paso por delante del atacante. La lista de correo de bugtraq es un gran recurso a este respecto.

Introducción a NMAP

A lo largo de todo el texto hemos ido viendo poco a poco algunas de las funciones de nmap, pudiendo comprobar que se trata de una de las herramientas de análisis de red más polivalentes que existen. En mi opinión, es **el mejor escaneador de puertos y sistema de detección de fingerprint** que existe hoy en día, y además es **software libre**.

Como ya hemos visto las distintas **opciones de escaneo de nmap**, no vamos a volver a hablar de ellas, pues tanto la parte técnica como los comandos concretos de nmap ya han sido vistos. Vamos a centrarnos en **el resto de funcionalidades de nmap** y a ver algunos ejemplos.

Verbose: Habréis visto que siempre que utilizo el comando nmap añadía dos parámetros: **-P0** y **-vv**. Ya sabemos que **-P0 sirve para evitar que hagamos ping al host**, así que nos queda saber para qué sirve **-vv**. Se trata del **modo detallado (verbose)** de nmap, para obtener por pantalla **información adicional** acerca de las acciones llevadas a cabo y sus resultados. Usándolo **una vez (-v)** obtendremos **información adicional**, usándolo **dos veces (-vv)** obtendremos **más aún**, y **usando -bb** obtendremos más aún (según el manual de nmap, **nos volveremos locos haciendo scroll** en pantalla...).

Uso: ***nmap*** *-P0* ***-vv*** *-sS xxx.xxx.xxx.xxx*

Rango de puertos: No siempre querremos escanear los 65535 puertos de un host... y mediante el parámetro -p podremos **establecer el rango** que queremos escanear y ahorrarnos tiempo. Por defecto, nmap escanea el **rango 1-1024** y además los que se encuentran definidos en ***/etc/services***.

Uso para un puerto (80): ***nmap*** *-P0 -vv -sS* ***-p 80*** *xxx.xxx.xxx.xxx*

Uso para un rango (1-1024): ***nmap*** *-P0 -vv -sS* ***-p 1-1024*** *xxx.xxx.xxx.xxx*

Puerto de origen: Si estamos detrás de un firewall o algún dispositivo que bloquea la salida de información desde determinados puertos, podemos especificar qué puerto queremos usar para lanzar el escaneo.

Uso (6969): ***nmap*** *-P0 -vv -sS* ***-g 6969*** *xxx.xxx.xxx.xxx*

Número máximo de sockets: Cuando usamos el escaneo TCP connect(), podemos definir el número máximo de sockets a abrir con el fin de no colapsar el host que queremos escanear.

Uso (7): ***nmap*** *-P0 -vv* ***-sT -M 7*** *xxx.xxx.xxx.xxx*

Modo rápido: Si únicamente queremos escanear los servicios listados en el fichero ***/etc/services*** debemos usar la opción -F.

Uso: ***nmap*** *-P0 -vv -sS* ***-F*** *xxx.xxx.xxx.xxx*

Uso de señuelos: Esta funcionalidad es una delicia. Mediante el uso de una serie de **señuelos**, podemos hacer que nmap mande, además de los paquetes del escaneo en cuestión, una serie de **paquetes falsificados (packet spoofing)** para que parezcan provenir de otros hosts. De esta forma, se generarán en los logs una **lista de hosts** que han realizado el escaneo de puertos y **no se sabrá cuál de ellos ha sido el verdadero autor**. El orden a la hora de colocar los señuelos y nosotros mismos (ME) es indiferente, si bien conviene que nuestro propio host **no sea de los primeros**, con lo que evitaremos que ciertos firewalls siquiera logeen nuestra IP. Es importante asegurarse de que los señuelos usados **están online** para **evitar causar un SYN flood** en la víctima (o... es importante asegurarse de que estén offline para provocar un SYN flood...).

Uso: ***nmap*** *-P0 -vv -sS* ***-Dsñ1.sñ1.sñ1.sñ1,sñ2.sñ2.sñ2.sñ2,[...],ME,[...],sñn.sñn.sñn.sñn*** *xxx.xxx.xxx.xxx*

Uso de IPv6: Es posible usar nmap con el protocolo IPv6 en lugar de IPv4.

Uso: ***nmap*** *-P0 -vv -sS* ***-6*** *xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx*

Política de precaución: Mediante esta opción **(-T)** establecemos el **retardo** entre unos paquetes y otros. Cuanto **más rápido** los mandemos, **antes terminará** el escaneo pero **más llamativo resultará**; y cuanto **más lento** lo realicemos, **más tardará en terminar**, pero **más difícil de detectar** resultará. Por defecto se usa normal.

Uso (Paranoico): ***nmap*** *-P0 -vv -sS* ***-T Paranoid*** *xxx.xxx.xxx.xxx*

Uso (Furtivo): ***nmap*** *-P0 -vv -sS* ***-T Sneaky*** *xxx.xxx.xxx.xxx*

Uso (Educado): ***nmap*** *-P0 -vv -sS* ***-T Polite*** *xxx.xxx.xxx.xxx*

Uso (Normal): ***nmap*** *-P0 -vv -sS* ***-T Normal*** *xxx.xxx.xxx.xxx*

Uso (Agresivo): ***nmap*** *-P0 -vv -sS* ***-T Agressive*** *xxx.xxx.xxx.xxx*

Uso (Demente): ***nmap*** *-P0 -vv -sS* ***-T Insane*** *xxx.xxx.xxx.xxx*

Resolución de DNS: A veces nos interesará que se resuelvan los nombres de dominio y otras que no se haga. Por defecto se realiza a veces, dependiendo de la circunstancia.

Uso (no resolver): ***nmap*** *-P0 -vv -sS* ***-n*** *xxx.xxx.xxx.xxx*

Uso (resolver siempre): ***nmap*** *-P0 -vv -sS* ***-R*** *xxx.xxx.xxx.xxx*

Archivo de registro: Podemos guardar un registro de la salida por pantalla en un fichero de varios tipos, según la necesidad.

Uso (texto plano): ***nmap*** *-P0 -vv -sS* ***-oN <fichero.algo>*** *xxx.xxx.xxx.xxx*

Uso (XML): ***nmap*** *-P0 -vv -sS* ***-oX <fichero.algo>*** *xxx.xxx.xxx.xxx*

Uso (tratable por grep): ***nmap*** *-P0 -vv -sS* ***-oG <fichero.algo>*** *xxx.xxx.xxx.xxx*

IP e interfaz usada: En caso de que haya varias interfaces, o en caso de que nmap no la autodetecte, se puede especificar ésta a mano.

Uso (IP): ***nmap*** *-P0 -vv -sS* ***-S IP.IP.IP.IP*** *xxx.xxx.xxx.xxx*

Uso (interfaz): ***nmap*** *-P0 -vv -sS* ***-e <intX>*** *xxx.xxx.xxx.xxx*

Modo interactivo: Arranca nmap como modo interactivo (una vez arrancado, pulsando h obtenemos ayuda más detallada).

Uso: ***nmap --interactive***

Especificación de objetivos: No siempre querremos escanear una única IP, o puede que no queramos escribirla cada vez, o que la tengamos en un fichero de logs...

Uso (Entrada por fichero): ***nmap*** *-P0 -vv -sS* ***-iL <fichero.algo>***

Uso (Rango con notación de máscara): ***nmap*** *-P0 -vv -sS* ***xxx.xxx.xxx.xxx/xx***

Uso (Rango personalizado): ***nmap*** *-P0 -vv -sS* ***'xxx.xxx.xxx.xxx-yyy.yyy.yyy.yyy'***

Uso (Rango personalizado con asteriscos): ***nmap*** *-P0 -vv -sS* ***xy*.xy*.xy*.xy****

Software

Además de nmap, existen muchos otros programas interesantes orientados al escaneo de puertos y/o a la detección del fingerprint de un sistema operativo. Vamos a resumir los más interesantes (por orden alfabético):

7th Sphere Portscan
Escáner de puertos
Última versión: 1.2
Sistema: Windows
Licencia: Freeware
Descarga: http://www.zone-h.org/download/file=3982/

Cheops (http://www.marko.net/cheops/)
Escáner de puertos y fingerprint
Última versión: 0.61
Sistema: Unix
Licencia: GPL
Descarga: ftp://ftp.marko.net/pub/cheops/cheops-0.61.tar.gz

EssentialNetTools (http://www.tamos.com/products/nettools/)
Escáner de puertos y vulnerabilidades
Última versión: 3.2
Sistema: Windows
Licencia: Copyright
Descarga: http://www.webattack.com/dlnow/rdir.dll?id=101212

GNU Netcat (http://netcat.sourceforge.net/)
Escáner de puertos
Última versión: 0.7.1
Sistema: Unix
Licencia: GPL
Descarga: http://cesnet.dl.sourceforge.net/sourceforge/netcat/netcat-0.7.1.tar.gz

IP-Tools (http://www.ks-soft.net/ip-tools.eng/)
Escáner de puertos y vulnerabilidades
Última versión: 2.30
Sistema: Windows:
Licencia: Copyright
Descarga: http://hostmonitor.biz/download/ip-tools.exe

LANguard (http://www.gfi.com/languard/)
Escáner de puertos y vulnerabilidades
Última versión: 5.0
Sistema: Windows
Licencia: Copyright
Descarga: http://www.gfi.com/lannetscan/networkscanner_freeware.htm

MegaPing (http://www.magnetosoft.com/products/megaping/megaping_features.htm)
Escáner de puertos
Última versión: 4.3
Sistema: Windows
Licencia: Copyright
Descarga: http://www.magnetosoft.com/downloads/MegaPingSetup.exe

Nessus (http://www.nessus.org/)
Escáner de puertos y vulnerabilidades
Última versión: 2.0.12
Sistema: Unix
Licencia: GPL
Descarga: ftp://ftp.gwdg.de/pub/linux/misc/nessus/nessus-2.0.12/src/

Netcat (http://www.atstake.com/research/tools/network_utilities/)
Escáner de puertos (puede servir también :-P)
Última versión: 1.10
Sistema: Unix & Windows
Licencia: Freeware
Descarga (Unix): http://www.atstake.com/research/tools/network_utilities/nc110.tgz
Descarga (Windows): http://www.atstake.com/research/tools/network_utilities/nc11nt.zip

NeWT (http://www.tenablesecurity.com/newt.html)
Escáner de puertos y vulnerabilidades
Última versión: 2.0
Sistema: Windows
Licencia: GPL
Descarga: http://cgi.tenablesecurity.com/tenable/requestForm.php

NMAP (http://www.insecure.org/nmap/)
Escáner de puertos y de fingerprint
Última versión: 3.55
Sistema: Unix
Licencia: GPL
Descarga: http://download.insecure.org/nmap/dist/nmap-3.55.tar.bz2

NMAP for Windows (http://www.insecure.org/nmap/)
Escáner de puertos y de fingerprint
Última versión: 1.3.1
Sistema: Windows
Licencia: GPL
Descarga: http://download.insecure.org/nmap/dist/nmapwin_1.3.1.exe

p0f (http://freshmeat.net/projects/p0f/)
Escáner de fingerprint pasivo
Última versión: 2.04
Sistema: Unix
Licencia: GPL
Descarga: http://freshmeat.net/redir/p0f/43121/url_tgz/p0f-2.0.4.tgz

QueSO
Escáner de fingerprint
Última versión: 0.980922b
Sistema: Unix
Licencia: GPL
Descarga: http://ftp.debian.org/debian/pool/main/q/queso/queso_0.980922b.orig.tar.gz

Retina Network Security Scanner (http://www.eeye.com/html/products/retina/index.html)
Escáner de puertos y vulnerabilidades
Última versión: No proporcionada en la web del fabricante
Sistema: Windows
Licencia: Copyright
Descarga: http://www.eeye.com/html/products/retina/download/index.html

Shadow Security Scanner (http://www.safety-lab.com/en/products/1.htm)
Escáner de puertos y vulnerabilidades
Última versión: 7.01
Sistema: Windows
Licencia: Copyright
Descarga: http://www.safety-lab.com/cgi/download.pl?lang=en&prog=ShadowSecurityScanner

Siphon (http://siphon.datanerds.net/)
Escáner de fingerprint
Última versión: 0.666
Sistema: Unix
Licencia: GPL
Descarga: http://siphon.datanerds.net/siphon-v.666.tar.gz

WinFingerprint (http://winfingerprint.sourceforge.net/)
Escáner de fingerprint
Última versión: 0.5.11
Sistema: Windows
Licencia: GPL
Descarga: http://heanet.dl.sourceforge.net/sourceforge/winfingerprint/winfingerprint-0.5.11a.zip

Xprobe2 (http://www.sys-security.com/html/projects/X.html)
Escáner de fingerprint
Última versión: 0.2
Sistema: Unix
Licencia: GPL
Descarga: http://www.sys-security.com/archive/tools/xprobe2/xprobe2-0.2.tar.gz

Distribución de este documento

Este documento se distribuye en formato PDF realizado bajo OpenOffice.org 1.1.2.

Ficheros a distribuir:

Nombre: “Sabuesos.pdf”
Descripción: Documento principal.

Nombre: “Sabuesos.pdf.sig”
Descripción: Firma digital PGP del fichero “Sabuesos.pdf” realizada por el autor.

Nombre: “hash.txt”
Descripción: Contiene la cadena hash MD5 de los ficheros “Sabuesos .pdf” y “Sabuesos .pdf.sig”.

Datos adicionales:

El hash MD5 puede resultar útil para comprobar la integridad del fichero descargado, pero no es garantía de la inalterabilidad del documento, pues puede haber sido alterado junto a la cadena de hash.

Para comprobar la completa autenticidad e inalterabilidad del fichero, usar el sistema PGP para validar el fichero .sig (MIME/PGP) de firma. Cualquier modificación no autorizada del documento hará que la firma del mismo no sea válida, y ésta es imposible de falsificar.

Death Master

Autentificación:

-----BEGIN PGP MESSAGE-----
qANQR1DBwUwDJoT5ygJgu7ABD/9Gzhy7eDexRQKws0DTXJXPSOk/Fa7V4yHNUgY8
f7zJ1s5kd0CGA5pHZGKl6bGdtdjcjNJ/0ZLMereu1GViJevhICTjQ3kO0OPlOIcz
kfA55a5Zus0WjVdZb2L3CeYlG+7Ndu5V7ly9TCe+//lUVjEaPbFo/EjUbVCRYqFG
FJKTSUT/PcHFYTowGHtPuCCTFmLm3YUd/TS2CFPmDoVaWOl9Z24tsSQqVVNcEZgk
wJo4SDBlItzAzBE7GOQN601ctBoCe1zeI6JeEYz+TsUzuMekJHgawvln7rWXH8Z9
Ez4CbvMfwFze1zo816M6rKrIMdlvyaz36Vtc6LmV481FJ2vSeCyGRycagcIk5OBC
HvE5VhX5c7XlES/40iRKuOkeAusvlXGANZmc8qDTkidHOvrXIKT3ANdca7WVJTl3
SLzmZl9uN0ePSWlXJE06jk5z9zZwFBM4ylf1Q0P6SxjT8UzHw+uoOOLHGeqK/5vF
T3boL9VkbaQi2zir4WByqUze6nYPa0S4N+79eD9NbU5kW1dbY3d48gCKDhg4PVEN
aBtjv5noELvvNfofJ1xm4KGfrSDvvxje3GKhJCl7azKxZqpQAGJlq+LPBb6Mq1GQ
jTU7XuPwqb8dJ+xaRe25HCXBlUO1BNlDz2R1nv9hC0lavqIF3T/H7jP8Eh5JTk/B
j32xB9LCjwE6NOgZLW5iDHuk2F3tZUfV0+7CfirVVSZUZlOXd55A11rCXpgaYfb7
+pWNYTyKbq0fQ4fMWoraL0KFzqTojoQbrIe6tkJo41q7TO/yl7ob4AByVQ3r/oY2
vVkEyBKytn/8VxSxvx09Cwk8LLw6NwH2eF1H27TaK5Yls3c7ZfclqfC1EGG66cpK
ujcCWGNeo+iXblksm8lnPD0eRnaSyVrqrdpInGE+9sEASVmNrpPIYiWET4b5Ht6j
geeMJXCgEguMwbdYhjdWJ9UM4iFDCokosS/fXm0oykWSFjwwemeiWs3m0n3I+aOj
5oNaKwyywpPr0jrdCZigHtf98JDVpkQeMmMwEKhbuXY9GpAXCreTJAskUq8gFuqe
kTkO/7brcCp83lt1FRFLrcaGa2NwdBNKc6cTeqWCF/xIQ+9Yj4i4s3M3nsNslWbn
1SFhO4hoRqE0U1fZh2trkdRFco7qrsprbw9jvskaY3Da3CK4L0kAmEo4YAEk8CPi
MNOMTcprUZ71F3y8seLO7fVioUOiLdAWtteyg63aS9qLlkfZzlInFW1T2L4s1I/l
9lYnTze8HDuQGFjR1PEFuw2YJvg5GAcNPcW5qMVdqvXDczzkfi7H0XkT5taHdbgR
bHpq8hwFZ5YTC7rQuXm49cXtCz95wExzmSuNzTU44+l3bw1qZ+sxuVSbyRVtZ4BQ
pVfPcz9FBUVtix18KX3Ip+YpnsSyPnneOEUif5tGV4Xt0trTzWYVLhPmetNLigts
gU+kkVmZ+u6+jqkfwZ2/BCIkDzf629SJxXIJLUmrHhURghN+fE7VM3jLXCuglLq1
hVo3bdCqPeq1TymrB8cM97/MwVCSQCZXdMb3WNaSSMoylWlh4C8+iIJfRJYDbEk9
7dRmu9K3UJXXNQhF+orqBtW50HVhwybM4uoqdsZ0k+kWA8Zcdzrw0sH4Q6oLUrsM
KHbbhhiTWCIwDH3W4DMjZUCniHweu8GgciJjtMwrpUmz7QNVDlb9TePmrNGHkvk6
lNaE65mWR8m176ffDZzZ7mQwKRnFxTDgnKkAqJLucFuP2H7hoTbUh4CnOyiQDv1z
JQL7rSfVSsHhIkRJBdKFQnHTJB4Z5CpezaNKLgumzOeKcMEJFF4=
=apCn
-----END PGP MESSAGE-----

Licencia

Sabuesos en la Red: El escaneo de puertos – http://www.death-master.tk/ – Versión 1.0

Este documento ha sido liberado por su autor bajo la licencia GNU Free Documentation License (GFDL), y su utilización, copia o reproducción queda sujeta a los términos de la citada licencia, que puede ser consultada en el siguiente sitio web:

- **GNU Free Documentation License:** http://www.gnu.org/copyleft/fdl.html

GFDL Version 1.2, November 2002
Copyright (C) 2000, 2001, 2002 Free Software Foundation, Inc.

Copyright (c) 2004 Death Master
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with the Invariant Sections being “Distribución de este documento” and “Licencia”, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Cualquier copia, modificación, distribución o utilización en general de este documento debe respetar la autoría original del mismo, correspondiente a **Death Master**.

Bloodhounds in the Net: The port scanning – http://www.death-master.tk/ – Version 1.0

This document has been freed by its author under the license GNU Free Documentation License (GFDL), and its use, copy or reproduction is subject to the terms of the mentioned license that can be consulted in the following website:

- **GNU Free Documentation License:** http://www.gnu.org/copyleft/fdl.html

GFDL Version 1.2, November 2002
Copyright (C) 2000, 2001, 2002 Free Software Foundation, Inc.

Copyright (c) 2004 Death Master
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with the Invariant Sections being “Distribución de este documento” and “Licencia”, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Any copy, modification, distribution or general purpose use of this document should respect the original responsibility of it, corresponding to **Death Master**.

*** End Of File ***