CURSO DE TCP/IP (2ª Entrega)

EL PROTOCOLO DE TRANSPORTE UDP (PROTOCOLO DE DATAGRAMAS DE USUARIO)

- Vamos a descubrir las herramientas necesarias para "crear" paquetes de red
- Empezaremos a desgranar las capas de red
- Trataremos el Protocolo de Datagramas de Usuario

1. INTRODUCCIÓN

Espero que no os perdieseis el primer artículo del curso de TCP/IP (Transmisión Control Protocol / Internet Protocol --- Protocolo de Control de Transmisión / Protocolo de Internet), porque voy a asumir que habéis comprendido a grandes rasgos el concepto de capas de protocolos, y el mecanismo básico de transporte de paquetes de datos a través de Internet (o cualquier otra red TCP/IP).

NOTA DE PC PASO A PASO / HXC

Para todos aquellos que no tienen la primera entrega del curso de TCP / IP publicada en el número 17 de PC PASO A PASO, hemos decidido pasarla a formato PDF y liberarla en la Web de la revista (www.hackxcrack.com).

También aprovechamos esta "nota" para indicar a nuestros lectores que todos los artículos liberados y los programas que mencionamos en los artículos están disponibles en la sección ARTÍCULOS LIBERADOS Y DESCARGAS de la Web.

Para esta primera "lección" he escogido un protocolo muy sencillo, el protocolo UDP (User Datagram Protocol --- Protocolo de Datagramas de Usuario), para así poder extenderme más en explicaros algunas herramientas que nos pueden ser útiles a lo largo de todo el curso. De momento, estas herramientas las utilizaremos para manejar el protocolo UDP, pero en futuras lecciones exprimiremos al máximo las herramientas para hacer maravillas con las demás capas de protocolos: TCP, IP, ARP (Address Resolution Protocol --- Protocolo de Resolución de Direcciones), etc.

Empezaré explicando a grandes rasgos el mecanismo de transporte de un paquete UDP a través de Internet para luego detallar las cabeceras UDP y, por último, veremos todo esto de forma práctica utilizando unas herramientas que nos permiten construir paquetes UDP desde cero (los llamados "raw sockets").

Pero, antes de empezar, he de aclararos un asunto. Supongo que algunos de vosotros os preguntareis qué ha pasado con la serie RAW. Mis intenciones no eran de ninguna manera sustituir la serie RAW por el curso de TCP/IP, si no desarrollar ambos en paralelo. Pero es increíble de qué manera pueden conspirar juntas las circunstancias de la vida para cambiar por completo tu rumbo cuando menos te lo esperas. En poco más de un mes me han surgido una serie de problemas de todo tipo (familiares, personales, de salud, laborales, y académicos) por los cuales ahora (y sin plazo definido) dispongo de muchísimo menos tiempo.

He barajado un poco mis posibilidades, y creo que lo que puedo permitirme de momento es continuar sólo con el curso de TCP/IP, aunque no descarto publicar algún otro artículo de la serie RAW algún mes que consiga juntar algo más de tiempo. Con esto lo que os quiero

decir es que no os prometo nada con respecto a la serie RAW, pero que haré lo que esté en mi mano para que no termine aquí, aunque haya que avanzar ahora mucho más despacio.

Gracias por vuestra comprensión. ;-)

NOTA EDITORIAL

Para los nuevos lectores, simplemente decir que la serie de artículos denominada "SERIE RAW" ha estado presente en casi todos los números de esta publicación y ha constituido uno de los pilares de la misma.

La Serie RAW ya nos ha aportado conocimientos sobre los Protocolos más importantes y desde hace mucho tiempo se nos ha reclamado el inicio de un curso de TCP / IP. Finalmente, desde el anterior número 17 por fin ese curso ha llegado.

Tal y como nos describe el autor, esperamos poder hacer más entregas de la Serie Raw en futuros números, pero de forma más dilatada en el tiempo. Mientras tanto disfrutemos de este excelente curso de TCP / IP, una serie de artículos ampliamente reclamada y esperada por nuestros lectores.

2. FUNCIONAMIENTO DEL PROTOCOLO UDP

Los que os perdisteis la introducción de este curso no tenéis que preocuparos, ya que este primer punto resume en cierto modo lo que expliqué entonces. Aún así, os recomiendo que leáis la introducción, ya que aquí voy a resumir en un par de páginas lo que allí explicaba en más de 20 páginas.

Para los que sí que leísteis la introducción, este punto creo que os puede ayudar a aclarar las ideas a modo de resumen.

A lo largo de la serie RAW os expliqué varios protocolos que funcionaban sobre TCP, pero sólo uno que funcionase sobre UDP: el protocolo DNS. Vamos a utilizar este protocolo como ejemplo para ver qué ocurre desde que un cliente solicita una consulta DNS a un servidor, hasta que éste cliente recibe la respuesta pertinente. Os aconsejo que leáis el artículo sobre DNS de la serie RAW, que se encuentra en el número 14 de la revista, aunque voy a empezar haciendo una introducción sobre este protocolo.

NOTA DE PC PASO A PASO / HXC:

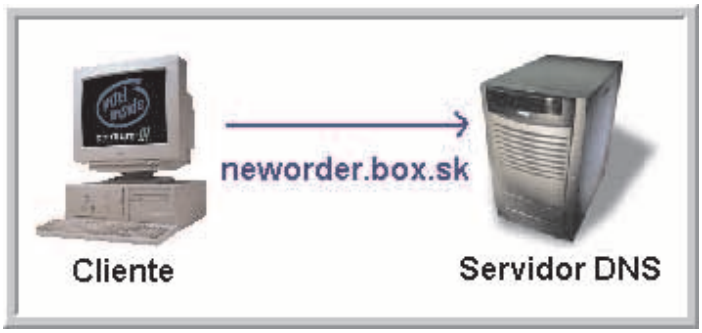
El artículo SERIE RAW: DNS mencionado por el autor ha sido liberado y está disponible en nuestra Web: www.hackxcrack.com

2.1. LAS CAPAS DE UDP/IP

Os recuerdo que el protocolo DNS es el que nos permite utilizar las famosas URLs en lugar de direcciones IP. Es decir, nos permite escribir en nuestro navegador http://www.google.com en lugar de tener que escribir http://216.239.39.99, que es la dirección IP de Google.

Para conseguir esto, existen repartidos por el mundo una serie de servidores encargados de traducir URLs a IPs, e IPs a URLs. Nuestros PCs se comunicarán con alguno de estos servidores cada vez que quieran utilizar una URL, para obtener así su IP correspondiente. Esta comunicación se lleva a cabo a través del protocolo DNS.

Cuando escribimos en nuestro navegador una URL, por ejemplo http://neworder.box.sk, nuestro sistema enviará una consulta DNS a un servidor, indicando en la consulta el nombre que desea traducir (en este caso neworder.box.sk, ya que el prefijo http:// es sólo una indicación al navegador sobre el protocolo a utilizar, pero no forma parte del nombre que deseamos traducir).



El servidor DNS nos enviará una respuesta que contendrá la IP correspondiente a ese nombre.

Gracias a esto, a continuación nuestro navegador podrá acceder a la máquina que contiene la página Web que deseamos visitar, ya que sólo puede existir una comunicación directa entre dos máquinas si cada una conoce la dirección IP de la otra.

Ahora vamos a pensar un poco en cómo se podría conseguir todo este mecanismo del DNS, en el cual un cliente solicita un nombre a un servidor, y el servidor le responde con la IP correspondiente. ¿Qué problemas se nos presentan a la hora de llevar a cabo este proceso aparentemente tan sencillo? Pensadlo un poco, y después mirad la lista de problemas a salvar que os pongo a continuación, para ver si habéis llegado a las mismas conclusiones:

- En primer lugar, por supuesto, hay que conseguir que ambas máquinas (cliente y servidor) tengan una **conexión física**, ya sea por cables, por radio, o por cualquier otro medio físico que les permita establecer una comunicación bidireccional.

- En segundo lugar, sabiendo que en Internet todas las máquinas están conectadas entre sí mediante una compleja red de cables y conexiones inalámbricas, es lógico pensar que será necesario conocer el camino a recorrer en toda esa maraña de cables para **enlazar ambas máquinas** entre sí.

- En tercer lugar, el cliente necesitará conocer la **dirección IP** del servidor DNS, ya que sólo conociendo la dirección IP de una máquina puedes acceder a ella a través de Internet.

- En cuarto lugar, tiene que existir algún mecanismo que le indique al servidor que la consulta que le estamos haciendo es una consulta DNS, y no de cualquier otro tipo. Por ejemplo, el servidor DNS podría ser al mismo tiempo un servidor Web, y un servidor de correo electrónico. Por tanto, tiene que existir un mecanismo que le permita distinguir qué clientes solicitan **servicios** DNS, cuáles solicitan servicios Web, y cuáles solicitan servicios de correo eletrónico.

A grandes rasgos, son cuatro los problemas que hemos encontrado para conseguir llevar a cabo esta comunicación. Y, por supuesto, no es coincidencia que sean cuatro las capas de protocolos utilizadas en una comunicación UDP: **capa física, capa de enlace, capa de red, y capa de transporte**.

Si no fuese gracias a la existencia de estas 4 capas diferenciadas, el protocolo DNS debería encargarse por sí sólo de solucionar todos estos problemas. Es decir, el protocolo DNS debería tener sus propias conexiones físicas entre máquinas, sus mecanismos para encontrar un camino entre todas las máquinas que están conectadas simultáneamente, sus propias direcciones IP, y sus propios mecanismos para diferenciarse de otros servicios (como la Web o el correo electrónico).

Esto convertiría al aparentemente sencillo protocolo DNS en un sistema de una complejidad inabarcable, y lo mismo ocurriría con cualquier otro protocolo que tuviese que

lidiar él solito con todos los problemas existentes en una comunicación.

Vamos a ver entonces cómo se reparte el trabajo de la comunicación para permitir que el protocolo DNS se abstraiga de todo lo que no sea su misión directa.

Empezamos escribiendo una url en nuestro navegador:
http://neworder.box.sk/home.html

En primer lugar, nuestro navegador quitará la parte de la URL que no corresponda al nombre, que es: **neworder.box.sk**.

Teniendo ya el nombre, nuestro sistema construye un paquete que contiene la consulta DNS.

Y ahí termina la responsabilidad del protocolo DNS, ya que su única función consiste en enviar consultas de nombres para recibir direcciones IP en respuesta. Por tanto, nuestro sistema pasará ahora la bola a otro protocolo, que será el encargado del **transporte** del paquete DNS.

Existen varios protocolos de transporte, aunque los más importantes son **TCP** y **UDP**.

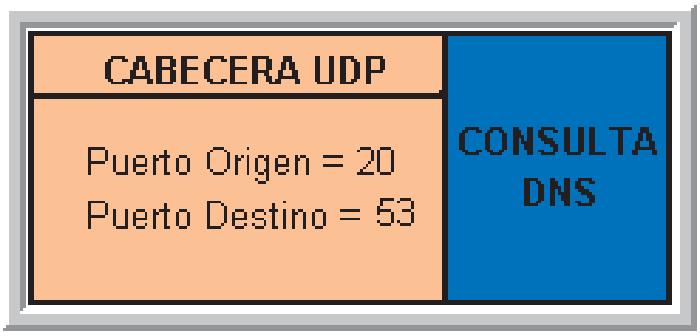
En este caso, el protocolo de transporte utilizado será **UDP**, así que el sistema pasará el paquete DNS al protocolo UDP para que éste realice su función, que consiste básicamente en marcar el paquete con un puerto de origen y otro de destino.

Según el puerto de destino asignado, la máquina que reciba el paquete (el servidor DNS) podrá saber a qué servicio está destinado.

En este caso, el **puerto de destino** es el **53** y, por tanto, es un paquete **DNS**. El **puerto de origen**, en cambio, servirá para que el servidor pueda responder a nuestra consulta, utilizando como puerto de destino el que para nosotros era un puerto de origen.

En resumen, lo que el protocolo UDP ha conseguido es identificar una comunicación entre dos máquinas, entre las cuales podría haber simultáneamente varias comunicaciones establecidas.

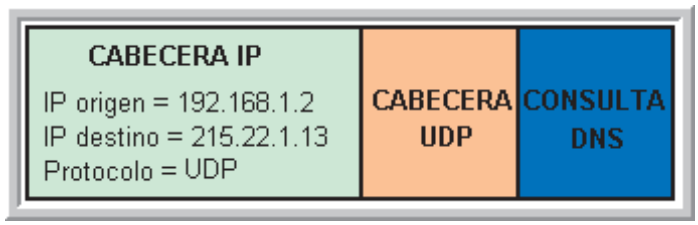
Lo que hace UDP es envolver el paquete con una pequeña cabecera que añade la información necesaria, es decir, los puertos.



El trabajo de UDP ya ha terminado, y tiene que pasar el paquete a otro protocolo para que se encargue del resto de problemas de comunicación. Concretamente, el próximo protocolo tendrá que solucionar el problema de identificar a dos máquinas (cliente y servidor) dentro de los millones que hay en toda la **red** Internet. En este caso, el protocolo de red utilizado será el protocolo **IP**.

Este protocolo se encargará de asignar las **direcciones IP** al paquete: la de origen será la nuestra, y la de destino será la del servidor DNS. Igual que sólo podemos llamar a una persona por teléfono si marcamos su número, sólo podremos acceder a una máquina de Internet si "marcamos" su dirección IP.

El protocolo IP, por tanto, añade al paquete una nueva cabecera, que contendrá las IPs de origen y de destino, así como una indicación de que el protocolo desde el cual le llegó el paquete fue el protocolo UDP.



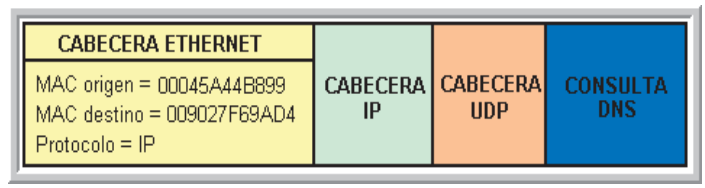
El paquete pasa ahora al protocolo **Ethernet**, un protocolo que permite encontrar el camino físico para **enlazar** dos máquinas (cada una con una dirección IP que ya conocemos gracias al protocolo IP).

Si cada máquina de Internet tuviese millones de cables para enlazarse con todas y cada una de las demás máquinas conectadas a la red, no sería necesario el uso de protocolos de enlace. Pero, al estar en Internet todo conectado con todo, necesitamos un mecanismo para encontrar el camino entre dos máquinas, aún conociendo las direcciones que las identifican (direcciones IP).

Exactamente lo mismo ocurre en el teléfono, lo cual podemos ver claramente si recordamos a las telefonistas de antaño. Antiguamente (y hoy también ocurre, pero con la diferencia de que está automatizado) en el instante en que marcabas un número de teléfono no se establecía una comunicación. Para conseguir esto hacía falta un procedimiento intermedio, que era el de las telefonistas. Las telefonistas se encargaban de ir conectando y desconectando cables en sus paneles para conseguir enlazar los dos puntos de la comunicación.

En el caso de Internet, a pesar de lo que os hayan contado vuestros padres, no existe una raza de enanitos encargada de conectar y desconectar cables, así que todo esto ocurre de forma automática y virtual, es decir, no se conecta ni se desconecta nada físicamente, si no que simplemente se establecen conexiones lógicas.

Para establecer estas conexiones se utilizan otro tipo de direcciones sobre las que ya hablé un poco en el artículo anterior, y hablaré mucho más en próximos artículos, por lo que de momento nos quedamos sólo con la idea de que el protocolo Ethernet añade su propia cabecera a todas las que había ya en el paquete.



Hasta ahora ya hemos solucionado los tres últimos problemas que mencioné (volved atrás y revisad la lista de problemas, y veréis que en efecto están solucionados), así que sólo nos queda el primer problema: la **conexión física**. La solución a este problema es bien sencilla: ¡enchufa el cable del módem, porque si no, por mucho protocolo que tengas seguro que no vas a poder enviar ninguna consulta DNS! :-P

Una vez que nuestro paquete está ya circulando por los cables físicos (u ondas de radio físicas), terminará llegando hasta el servidor DNS. El servidor analizará el paquete, verá que se trata de una consulta DNS, y hará lo que tenga que hacer para conseguir el resultado pedido (explicado en detalle en el artículo sobre DNS de la serie RAW, en el número 14 de la revista).

Una vez encontrada la respuesta, tendrá que construir un "sencillo" paquete de respuesta DNS que simplemente contendrá la dirección IP pedida como resultado de la traducción del nombre neworder.box.sk. Este paquete, para poder circular hasta nosotros de vuelta, también tendrá que tener sus propias cabeceras UDP, IP, y Ethernet.

El paquete completo (con las cabeceras) de respuesta podría ser parecido a este:

RESPUESTA DNS:
IP = 66.250.131.132

CABECERA UDP:
Puerto origen = 53
Puerto destino = 20

CABECERA IP:
IP origen = 215.22.1.13
IP destino = 217.12.1.15
Protocolo = UDP

CABECERA Ethernet:
MAC origen = 548F44A5558D
MAC destino = 54F566A47F88
Protocolo = IP

Os recomiendo que volváis atrás para ir comparando cada campo de la respuesta con los campos correspondientes en la consulta. Posiblemente veréis varias cosas que os parecerán muy extrañas. Por ejemplo, ¿por qué la IP de destino de la respuesta no es 192.168.1.2?Esto ya lo explicaré cuando hable sobre el protocolo IP a lo largo del curso. Y, ¿por qué ninguna de las dos direcciones MAC tiene nada que ver con las que había en la consulta? ¡Un poco de paciencia, por favor! ¡Que ya explicaré todo esto a lo largo del curso! :-)

2.2. FUNCIONES DE LA CAPA DE TRANSPORTE UDP

2.2.1. Identificación de conexiones

Lo importante que tenemos que comprender ahora, una vez hecho este repaso al sistema de protocolos por capas UDP/IP, es que gracias al protocolo UDP se puede identificar una comunicación específica entre dos máquinas, gracias a los puertos de origen y de destino.

Esto nos permite tener varias conexiones simultáneas entre dos máquinas (aunque las direcciones IP de cada conexión sean las mismas, las conexiones se pueden diferenciar gracias a los puertos), y también nos permite acceder al mismo servicio en varias máquinas diferentes (podemos, por ejemplo, estar viendo dos páginas Web de dos servidores diferentes al mismo tiempo).

Supongo que la necesidad de especificar un puerto de destino en cada paquete os habrá quedado muy clara, pero quizá no tenéis tan clara la necesidad de especificar también un puerto de origen.

Imaginemos que nuestro navegador realiza una consulta DNS con nuestro servidor DNS, pero casi al mismo tiempo nuestro cliente de correo electrónico también está realizando otra consulta DNS al mismo servidor. No podemos saber de ninguna manera cuál de las dos respuestas nos llegará antes.

Por una parte, si leísteis el artículo sobre DNS de la serie RAW, sabréis que el tiempo para procesar una consulta DNS puede variar enormemente. Por ejemplo, una traducción que se encuentre en la cache del servidor DNS nos llegará casi al instante, mientras que otra traducción menos afortunada podría requerir establecer varias conexiones con varias máquinas, desde los servidores raíz, y los servidores del TLD, hasta el último servidor de la jerarquía DNS. Si no sabéis de qué hablo no os asustéis, que esto no tiene nada que ver con UDP, si no con DNS, y no os hace falta conocerlo ahora. :-)

La idea con la que os tenéis que quedar es que de ninguna manera se puede asumir que al hacer dos consultas una detrás de otra, las dos respuestas nos llegarán en el mismo orden. Por tanto, si el servidor nos envía las dos respuestas, necesitamos de alguna manera saber cuál de las dos es la que solicitó nuestro navegador, y cuál es la que solicitó nuestro cliente de correo electrónico. ¡De ahí la necesidad del puerto de origen!

Por supuesto, si leísteis el artículo sobre DNS os estaréis preguntando: ¿y qué hay de los transaction ID? ¡Permiten perfectamente diferenciar una consulta DNS de cualquier otra! Pues sí, es cierto, pero el caso de DNS es un poco especial, ya que no es normal que los protocolos que utilicen UDP (o TCP) para el transporte incluyan sus propios identificadores para cada conexión. Si se hizo así en el caso de DNS no fue por motivos de identificar cada conexión, ya que para eso

sobraba con lo que hace UDP, si no por motivos de seguridad, como ya vimos al hablar en la serie RAW sobre los ataques por envenenamiento de DNS.

En la práctica, hay casos en los que el puerto de origen es irrelevante, por lo que no siempre es obligatorio especificarlo. En estos casos, lo que se hace es usar el puerto 0 como origen.

2.2.2. Corrección en los datos

Imaginad que por cualquier problema en la "línea" los datos llegasen corruptos hasta nosotros, y uno de los números que forman la IP que hemos solicitado al servidor DNS es incorrecto.

Sería un gran problema no tener una cierta seguridad de que los datos que nos llegan de un servidor DNS son correctos. Estaríamos cada dos por tres estableciendo conexiones con direcciones IP incorrectas.

UDP no puede garantizar que los datos lleguen, pero si que nos da cierta garantía de que, si han llegado, son correctos. Para ello incorpora en su cabecera un dato que no hemos visto antes (ya que estabamos viéndolo sólo a grandes rasgos), que nos permite comprobar que los datos son correctos, tal y como veremos más adelante.

Este problema de la corrección de los datos perfectamente podría haberlo incluido en la lista de problemas de la comunicación, pero no lo hice a propósito para que saliesen justo cuatro. :-P

2.3. LO QUE NO NOS DA EL PROTOCOLO UDP FRENTE A TCP

2.3.1. Fiabilidad en la comunicación

Aunque antes desmentí el mito de los enanitos telefonistas de Internet, unos enanos que sin duda si que existen en la red son los enanos gruñones que se dedican a desconectar cables de vez en cuando, ocasionando así la pérdida de paquetes.

No cabe duda de que ninguna tecnología es perfecta, y por eso siempre puede haber pérdidas de datos en cualquier comunicación. ¿Qué ocurriría, por ejemplo, si no nos llegase la respuesta del servidor DNS, a pesar de que éste la hubiera enviado? Si, por cualquier motivo, la respuesta se perdiese por el camino, no pudiendo llegar hasta nosotros, ¿habría alguna forma de detectar y solventar esta situación?

Analizando el funcionamiento básico del protocolo UDP, tal y como lo hemos visto, no habría ninguna manera. En UDP cada máquina envía sus paquetes a la red, sin tener ninguna certeza de si llegarán o no a su destino. Una vez que el servidor DNS envíe su respuesta, se desentenderá del asunto, al no tener forma de saber si la respuesta nos ha llegado o no.

Este es el principal problema de los protocolos **no orientados a conexión**, como es el caso de UDP. Como ya vimos en el artículo sobre DNS, en el caso de TCP esto es muy diferente, ya que TCP es un protocolo **orientado a conexión**. Por cada paquete transmitido en TCP es obligatorio que el receptor envíe un acuse de recibo para que el emisor sepa con certeza que los datos han llegado al destino. Esto **no** es así en UDP.

En UDP no hay manera de saber si los datos llegan al destino. Esto es una gran desventaja pero, por otra parte, tiene la gran ventaja de hacer la comunicación mucho más fluida, al no verse interrumpida constantemente por miles de paquetes de acuse de recibo.

Esta característica convierte a UDP en un protocolo de transporte ideal para comunicaciones sencillas (como DNS, o TFTP (Trivial File Transfer Protocol --- Protocolo Trivial de Transferencia de Ficheros)), y para envíos masivos de flujos de datos (como en

transmisiones multimedia de vídeo, audio, ...), pero en un protocolo que de ninguna manera sirve para comunicaciones que requieren fiabilidad (FTP, IRC, HTTP, ...).

Por ejemplo, ahora que menciono el IRC, imaginemos lo que sería una conversación en un chat en el que no tuviésemos la certeza de que lo que decimos vaya a llegar al resto de interlocutores. Algunas frases se perderían por el camino, y nadie podría saberlo, por lo que muchas conversaciones perderían su sentido y, probablemente, se convertirían en diálogos de besugos (aunque, incluso utilizando TCP hay gran cantidad de diálogos de besugos en IRC, pero eso más que un problema tecnológico es un problema intelectual).

2.3.2. Flujo ordenado de datos

Otro gran problema potencial de UDP es el del **desorden**. En el caso de DNS parece todo muy sencillo: una consulta -> una respuesta. Pero, ¿qué ocurriría si, por ejemplo, intentásemos hacer un chat mediante UDP?

Internet es una red increíblemente compleja, y existen muchos factores que pueden determinar que los paquetes vayan más o menos rápidos en cada instante. Esto puede dar lugar a situaciones tan extrañas como que dos paquetes salgan uno tras otro de una máquina, pero primero llegue al receptor el último que salió. Si no hay un mecanismo para controlar el orden en que deben ir los paquetes, en el chat veríamos respuestas a preguntas aún no formuladas, monólogos sin sentido, etc.

Otros protocolos de transporte, como TCP, sí que incluyen mecanismos para garantizar el correcto orden de los paquetes; pero no es el caso de UDP, por lo que es otro aspecto a tener en cuenta a la hora de decidir si nuestra aplicación funcionará mejor sobre un protocolo seguro y robusto como es TCP, o sobre un protocolo más fluido como es UDP.

2.3.3.Tratamiento adecuado de los paquetes grandes

Imaginemos que enviamos una consulta de DNS a un servidor, y éste nos responde, pero su respuesta se ha perdido por el camino. En una situación normal, tras pasar un tiempo sin recibir la respuesta, volveríamos a solicitarla, asumiendo que se perdió por el camino. ¡Vaya! Eso sí que es genial. Acabamos de solucionar el problema de la fiabilidad en la comunicación. ¿O no?...

¿Que ocurriría si en lugar de ser una simple consulta DNS nuestro paquete fuese una solicitud a un servidor FTP para bajarnos un archivo grande (por ejemplo una ISO de 700MB?

Si se perdiese el paquete de respuesta, el servidor tendría que volver a enviar nada menos que 700MB! Esto, por supuesto, es una barbaridad, y lo ideal para minimizar este tipo de problemas es dividir los paquetes demasiado grandes en paquetes más pequeños, para que luego estos se unan en el orden adecuado al llegar al destino, y reconstruir así el paquete original. Si ha habido algún fallo tecnológico en un momento dado, normalmente sólo se perderá alguno de los paquetes pequeños, que podrá ser reenviado sin problemas.

UDP no incorpora ningún mecanismo para partir los paquetes grandes, por lo que tampoco es adecuado para esta clase de comunicaciones. Quizá ahora os estaréis preguntando por qué entonces he dicho que UDP puede ser adecuado para transmisiones multimedia donde, sin duda, el flujo de datos es enorme.

La cuestión fundamental aquí es que en los contenidos multimedia no es crítica la pérdida de los datos. Por ejemplo, podemos estar transmitiendo una película a razón de 20 fotogramas por segundo, enviando por ejemplo cada fotograma en un paquete independiente. En este caso, la pérdida de un único paquete

no sería en absoluto crítica. Simplemente perderíamos un fotograma, y eso sería prácticamente imperceptible para el ojo humano, y para nada afectaría al espectador de la película. En cualquier caso, nunca sería necesario reenviar el paquete.

El tema de la partición de paquetes grandes es mucho más complejo de lo que puede parecer en un principio ya que, entre otras cosas, no sólo es misión del protocolo de nivel de transporte, pero mejor que me quede calladito ahora, que si no os voy a liar bastante, y ya habrá tiempo de ver todo eso a lo largo del curso. ;-)

3. UDP EN DETALLE

Antes de deciros cuál es el RFC que detalla el protocolo UDP os tengo que poner sobre aviso: podéis asustaros del tamaño del RFC, así que no tengáis miedo, que yo haré todo lo posible por resumiros lo más importante, como siempre. ;-)

El RFC en cuestión es el **RFC 768** (ftp://ftp.rfc-editor.org/in-notes/rfc768.txt). Ejem... esto.... ¿sólo dos páginas? Bueno, quizá he exagerado un pelín con eso del tamaño. 0;-)

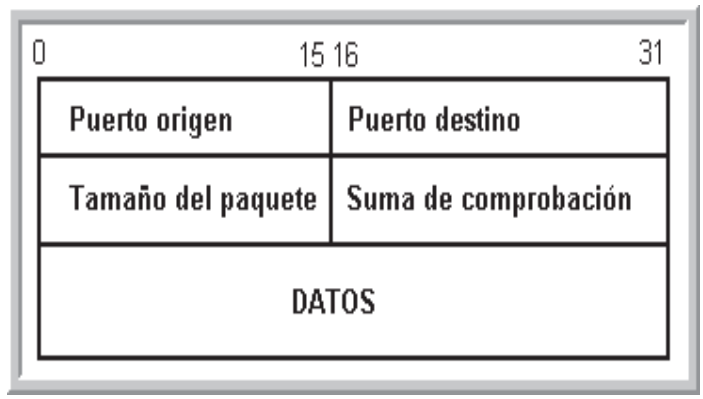
NOTA DE PC PASO A PASO / HXC

Para los que nunca nos han leido, decir que RFC (**R**equest **F**or **C**omments) son una serie de más de 3000 documentos donde se detalla todo lo relacionado con la tecnología de Internet.

Para los ALERGICOS al inglés, tienes al final de este artículo el RFC en perfecto castellano ;)

El RFC de UDP está sin duda escrito para gente entendida en el tema, por lo que no se andan con explicaciones, van directamente al grano, que es la especificación de las cabeceras UDP.

Como ya hemos explicado los conceptos, podemos permitirnos el lujo de poner las cabeceras sin explicar nada, como hace el RFC :)



Esta es la estructura de una cabecera UDP, es decir, del trozo que se añade a cada paquete para luego pasarlo al protocolo IP, el cual a su vez añadirá otro trozo, que se sumará también al trozo que luego añadirá el protocolo Ethernet.

En los ejemplos anteriores veíamos sólo dos campos: puerto de origen, y puerto de destino. Aquí no sólo vemos todos los campos, si no que además vemos su longitud.

Los "numerajos" que he puesto en la ilustración encima de la cabecera son el número de bits. Es decir, el campo **Puerto origen** abarca desde el bit 0 hasta el bit 15, es decir, 16 bits, que son 2 bytes. Por tanto, como 2^{16} son **65536**, éste es el número de puertos que existen en UDP (y también en TCP, por cierto).

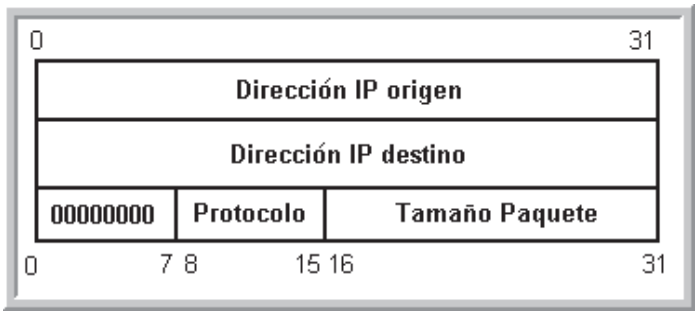
Como vemos, el campo **Puerto destino** abarca desde el bit 16 hasta el 31, que son otros 16 bits.

El campo **Tamaño paquete** son otros 16 bits, así como el campo **Suma de comprobación**.

En el campo **DATOS** es donde se encuentra todo lo que ha englobado el paquete UDP. Por ejemplo, en el caso de DNS, la consulta DNS estaría en el campo **DATOS** del paquete UDP. Su tamaño ahora puede ser cualquiera, y precisamente para eso sirve el campo **Tamaño paquete**.

Lo he llamado **Tamaño paquete** porque es un nombre más claro, pero su nombre original es **Length** (por si leéis algo sobre el tema y os líais). Este campo indica el número de bytes que ocupa todo el paquete UDP, incluyendo la cabecera. Al ser la cabecera de un tamaño fijo de 64 bits, es decir, 8 bytes, el campo **Length** será siempre como mínimo 8 (sería el caso de un paquete que no contenga datos).

Con respecto al campo **Suma de comprobación** (**checksum** en el original) hay bastante más que decir. Éste es el campo que nos permite comprobar que los datos que hemos recibido son correctos, es decir, que el paquete no ha tenido problemas que hayan corrompido los datos durante su transporte. Para calcular la Suma de comprobación se genera una cabecera con los siguientes campos:



Todos estos campos los conocemos, excepto el campo **Protocolo**. Cada protocolo existente tiene un número único asignado que le diferencia de los demás protocolos, y esto permite por ejemplo, propagar la información sobre a qué protocolo hay que pasar un paquete después de procesarlo, tal y como hemos visto en el ejemplo al principio de este artículo, en el cual la cabecera IP contenía un campo **Protocolo = UDP**, y la cabecera Ethernet contenía un campo **Protocolo = IP**.

Esta lista está mantenida por el **IANA** (**I**nternet **A**ssigned **N**umbers **A**uthority), así que podéis consultarla en www.iana.org. Si queréis una lista más sencilla, aunque no actualizada, la tenéis en el **RFC 1700** (http://www.rfc-editor.org/rfc/rfc1700.txt).

Si habéis echado un vistazo a la lista, habréis visto que el protocolo **UDP** tiene asignado el número **17**. Por tanto, en la cabecera que estamos tratando ahora habría que poner un 17 en el campo Protocolo.

Con todos estos campos se realiza una operación de aritmética binaria, que es la suma en complemento a uno de 16 bits de toda la cabecera. No voy a entrar en detalles de cómo se realiza esta operación, así que si tenéis curiosidad podéis consultar **el RFC 1071** o, si lo preferís, tenéis aquí directamente un código en C que calcula la suma de comprobación (checksum) para el protocolo TCP:
http://www.netfor2.com/tcpsum.htm.

Si queréis utilizar este código para el caso de UDP sólo tenéis que cambiar esta línea:

```
u16 prot_tcp=6;
```

Por esta otra:

```
u16 prot_tcp=17;
```

Aunque, si queréis que quede un poco más bonito, preocupaos de cambiar también el nombre de la variable para que se llame **prot_udp**. ;-)

Cuando recibimos un paquete UDP, nuestro sistema compara la suma de comprobación con los datos que conoce sobre el paquete (IP de origen, IP de destino, y tamaño de paquete UDP), y si algo no concuerda sabrá que ha habido algún problema y los datos del paquete no son válidos.

Un detalle curioso con respecto al complemento a uno (la forma de representación binaria utilizada en la suma de comprobación) es que existen dos formas de representar el cero: o bien que todos los dígitos sean 1, o bien que todos sean 0.

Esto nos es útil en nuestro caso, ya que lo que se hace es usar la primera representación del cero (todos los dígitos son 1) para las

sumas cuyo resultado son cero, y la otra representación del cero (todos los dígitos son 0) indica simplemente que ese paquete no incluye una suma de comprobación, por el motivo que sea (por ejemplo, si el protocolo no requiere una comprobación de los datos).

4. UDP EN LA PRÁCTICA

Aquí voy a presentar algunas herramientas que nos pueden ser útiles a lo largo de todo el curso de TCP/IP. Intentaré no discriminar a nadie, explicando herramientas tanto para Windows como para Linux.

Concretamente, las herramientas que vamos a utilizar nos permiten construir paquetes desde cero para varios protocolos diferentes (en este caso, el que nos interesa es UDP).

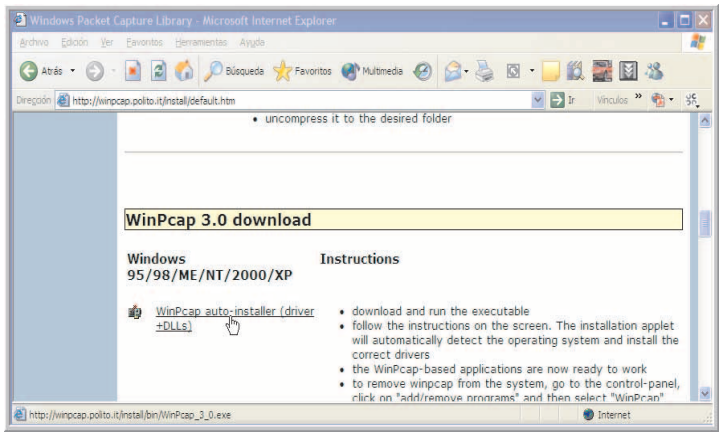
Para que estos programas funcionen, nuestro sistema tiene que permitir el uso de **raw sockets**, es decir, de paquetes generados desde cero. Linux siempre ha permitido los raw sockets, y Windows sólo lo hace desde sus versiones XP y 2000. Si aún queda algún usuario de Windows 9x (95, 98, o Millenium), que tampoco se preocupe, ya que puede utilizar las librerías **WinPCap** para que su sistema admita raw sockets. Estas librerías las tenéis en: http://winpcap.polito.it/.

De todas maneras, nosotros vamos a utilizar para nuestras prácticas en Windows un programita llamado **Nemesis.** Esta aplicación necesitará las librerías WinPCap incluso si tienes un Windows XP y 2000… tranquilos, ahora veremos detalladamente todo esto :)

4.1. NEMESIS PARA WINDOWS (TODAS LAS VERSIONES)

En primer lugar vamos con los usuarios de Windows, no porque sean más importantes, si no porque ya he empezado hablando de ellos antes. :-P

Lo primero que tenemos que hacer es bajarnos las librerías **WinPCap**. Para ello abriremos nuestro navegador preferido e iremos a http://winpcap.polito.it/, pincharemos en DOWNLOADS (en el menú de la izquierda) y aparecerá una lista descargas. Tenemos que descargarnos el WINPCAP 3.0

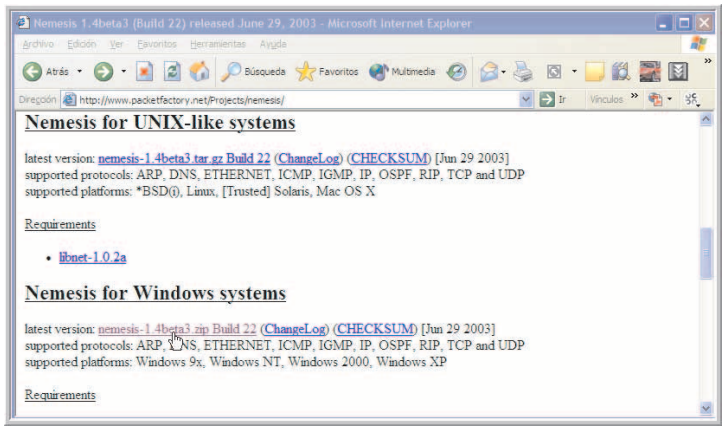


No cometas el error de bajarte la última versión (WINPCAP 3.1 BETA), hemos podido comprobar que no funciona correctamente en bastantes de nuestros equipos, incluso con configuraciones muy "normalitas". Así que ya sabes, descarga la versión 3.0 tal y como te indicamos :)

Una vez descargado el archivo mencionado (WinPcap_3_0.exe) lo ejecutaremos para que se instale en el sistema. Si te pide reiniciar Windows, hazle caso :)

A continuación, ya podemos bajar el **Nemesis**, que lo tenéis aquí:

http://www.packetfactory.net/Projects/nemesis/ (Por supuesto, bajad la versión para Windows).



Obtendremos el archivo comprimido nemesis-1.4beta3.zip y lo descomprimiremos en la carpeta c:\nemesis. Este programa no necesita instalarse, ahora veremos cómo utilizarlo ;)

Podemos pasar ya a la acción. Si vais al directorio (carpeta) sobre donde habéis descomprimido el Nemesis (en nuestro caso c:\nemesis), veréis que hay una serie de archivos **TXT**, uno de los cuales se llama **nemesis-udp.txt**. En ese archivo tenéis instrucciones para utilizar Nemesis con UDP, que es lo que nos interesa ahora. Enseguida descubriréis que Nemesis es una aplicación más práctica que estética, ya que no tiene interfaz gráfica, si no que funciona desde la consola **MS-DOS**.

Vamos a ver un primer ejemplo de uso de Nemesis:

Primero abrimos una consola de MS-DOS (una de nuestras habituales "ventanitas negras"). ¿Cómo se hace eso? Ya lo hemos explicado mil veces, tienes dos opciones:

- Menu Inicio --> Todos los Programas --> Accesorios --> Símbolo del sistema

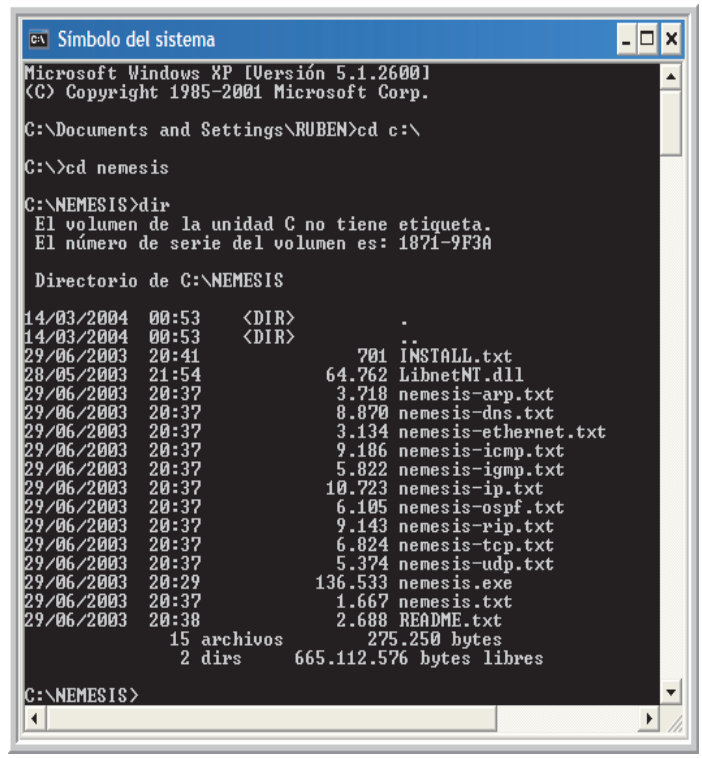
- Menu Inicio --> Ejecutar y en la ventana que nos aparezca escribimos command y pulsamos enter

Ahora, utilizando nuestra consola de MS-DOS vamos al directorio donde hemos tenemos en Nemesis esperándonos. Para el que no sepa hacer eso (que a estas alturas de la publicación ya deberían ser MUY POCOS):

- En nuestra ventanita negra escribimos **cd c:** y pulsamos enter (con esto nos vamos al directorio raíz del disco duro C).

- En nuestra ventanita negra escribiremos **cd c:\nemesis** y pulsaremos enter (con esto nos posicionamos en el directorio donde tenemos nuestro Nemesis).

- Escribimos **dir** y pulsamos enter. Con ello veremos un listado del contenido de la carpeta Nemesis, veremos, entre otros el archivo nemesis.exe (nuestro preciado Nemesis ;))



Por fin, escribimos:

nemesis udp –v –y 53 –D 194.224.52.6 –P –

y pulsamos enter, nos aparecerá algo como esto:

UDP Packet Injection -=- The NEMESIS Project Version 1.4beta3 (Build 22)

[IP] 68.253.120.0 > 194.224.52.6
[IP ID] 64988
[IP Proto] UDP (17)
[IP TTL] 255
[IP TOS] 00
[IP Frag offset] 0000
[IP Frag flags]

[UDP Ports] 64988 > 53

A continuación, escribimos cualquier cosa:

hola
Wrote 33 byte UDP packet.

UDP Packet Injected

Vamos a analizar lo que hemos hecho.

En primer lugar, al llamar a Nemesis con la opción **udp** en primer lugar (**nemesis udp**) le decimos que queremos enviar un paquete **UDP**, de entre todos los protocolos que soporta Nemesis.

La opción **–v** es la clásica opción **verbose** de muchos programas, que nos permite ver en pantalla información detallada sobre lo que estamos haciendo.

La opción **–y** sirve para detallar el **puerto UDP de destino**. En este caso, hemos utilizado el conocido puerto 53, que es el puerto de DNS.

La opción **–D** sirve para especificar la **dirección IP de destino**. En este caso, hemos utilizado la dirección de un servidor DNS de Telefónica.

Por último, la opción **–P** sirve para indicar qué datos queremos meter en el campo **DATOS** del paquete UDP. Por ejemplo, en este caso, debería ir aquí la supuesta consulta DNS. En el caso de este ejemplo, poniendo **–P –** indicamos a la aplicación que queremos meter nosotros los datos directamente desde teclado.

Analicemos ahora el comportamiento de Nemesis. En primer lugar, vemos que ha utilizado como IP de origen una que no es la nuestra, por lo que la respuesta a este paquete difícilmente podría llegarnos a nosotros. Es más, es también improbable que el paquete llegue siquiera a su destino, ya que los routers que habrá en los primeros pasos del camino (por ejemplo, nuestro router ADSL, o el router de nuestro ISP) probablemente rechazarán el paquete al no provenir supuestamente de una de las máquinas a las que tienen que dar servicio.

¿Y cual puede ser la utilidad de utilizar una IP que no es la tuya? Pues ya hablaremos sobre todo eso a lo largo del curso, pero os adelanto que esto puede servir para llevar a cabo gran número de ataques.

En nuestro caso no queremos atacar a nadie, simplemente queremos comprender de forma práctica el funcionamiento del protocolo UDP. Por lo tanto preferimos utilizar nuestra IP como IP de origen:

nemesis udp –v –y 53 –S 192.168.1.2 –D 194.224.52.6 –P –

Hemos añadido la opción **–S** para especificar la **IP de origen**. Si estamos detrás de un router, tendremos que especificar nuestra IP privada (de nuestra red local), ya que el router ya se encargará de convertirla luego en vuestra IP pública.

Ya se explico...

Ya se ha explicado mil veces, pero bueno… … para saber cuál es tu IP, simplemente abre una "ventanita negra", escribe **ipconfig /all** y pulsa enter. Este tema ya se trató en profundidad en los primeros números de Hack x Crack.

Si seguimos observando el comportamiento de Nemesis, veremos que ha escogido un puerto aleatorio como **puerto de origen**. Para la mayoría de las aplicaciones esto nos valdrá, pero si queremos probar a especificar un puerto de origen en concreto utilizaremos la opción **–x**:

nemesis udp –v –x 1000 –y 53 –S 192.168.1.2 –D 194.224.52.6 –P –

Esto puede sernos útil si estamos detrás de un firewall muy exigente que sólo nos permita utilizar algunos puertos UDP determinados (en el caso del ejemplo, el puerto 1000).

Por último, os habréis dado cuenta de que no es nada práctico meter los datos directamente desde teclado, ya que es bastante complicado generar una consulta DNS a pelo e introducirla desde el teclado. En la mayoría de los casos, os será mucho más útil construir primero la consulta con algún editor hexadecimal (por ejemplo, podéis utilizar UltraEdit, que además de ser un magnífico editor y visor de textos, es también un editor hexadecimal básico), guardarla en un **fichero**, y luego cargarla directamente en Nemesis con la opción **–P**. Por ejemplo, si nuestro fichero se llama **consulta.txt** haremos:

```
nemesis udp –v –x 1000 –y 53 –S 192.168.1.2 –D 194.224.52.6 –P consulta.txt
```

Podemos, por ejemplo, capturar una consulta DNS real con un sniffer, después modificarla a nuestra voluntad con el editor hexadecimal, guardar la consulta modificada en un archivo, y después enviarla a través de Nemesis.

En el artículo de la Serie RAW sobre DNS hablé acerca de la técnica de **envenenamiento de cache DNS**. Con Nemesis, y un sencillo shell script podríamos explotar esta técnica una vez conseguidos los datos necesarios. Si no habéis leído el artículo sobre DNS, os recomiendo que paséis este punto, porque probablemente no os enteraréis de nada. :-)

Si, por ejemplo, tenemos estos datos para llevar a cabo el ataque:

- **Dirección IP del servidor DNS de la víctima** = 194.224.52.6
- **Dirección IP del servidor DNS authoritative que queremos suplantar** = 194.220.51.2
- **Puerto UDP utilizado por el servidor DNS de la víctima** = 1200

Suponemos, por supuesto, que conocemos también el identificador de transacción, y que con él hemos construido una respuesta DNS falsa que tenemos almacenada en el archivo **envenenamiento.txt**. La forma de lanzar esta respuesta falsa sería:

```
nemesis udp –x 53 –y 1200 –S 194.220.51.2 –D 194.224.52.6 –P envenenamiento.txt
```

Podemos automatizar esto mediante un script que haga un bucle en el cual vaya utilizando distintos Transaction ID y, tal y como vimos en el artículo sobre DNS, según la versión de BIND que utilice el servidor DNS de la víctima tendremos una mayor o menor probabilidad de éxito.

4.2. HPING PARA LINUX

Existe también versión de Nemesis para Linux, pero os enseñaré mejor una herramienta bastante más potente, que es **Hping**. Podéis bajar Hping de www.hping.org. Para instalarlo tendréis que compilarlo primero. Los pasos a seguir son los típicos:

- Primero, una vez bajado el archivo con extensión **.tar.gz**, lo descomprimimos y desempaquetamos con el comando:
  tar -zxvf hping2.0.0-rc2.tar.gz

- Nos creará un directorio **hping2-rc2**, en el cual entraremos (**cd hping2-rc2**), para después ejecutar:
  ./configure

- A continuación, ejecutamos:
  Make

- Y, por último:
  make install

En caso de cualquier problema con este sistema estándar de instalación, podéis consultar el archivo **INSTALL** con instrucciones más detalladas sobre el proceso.

Una vez instalado, vamos a probar un poco su funcionamiento. Por supuesto, tendréis la correspondiente página del manual de hping:

```
man hping2
```

Hping no sólo permite enviar un único paquete,

si no que además implementa sencillos bucles para poder generar muchos paquetes sin necesidad de programar scripts. Si queremos que envíe un único paquete tendremos que usar la opción --count 1. Aunque mejor que veamos directamente un ejemplo:

hping2 193.224.52.6 --udp --destport 53 --file envenenamiento.dat --data 14 --count 1

El primer parámetro (**193.224.52.6**), como podéis imaginar, es la **IP de destino** del paquete, y es el único parámetro obligatorio.

El parámetro **--count** ya hemos dicho que indica el **número de paquetes** a enviar. Según las opciones estos paquetes podrán ser diferentes entre sí, por ejemplo, incrementando el puerto de origen en cada paquete. Si queréis más detalle sobre estas opciones consultad la página del manual. Por defecto, el puerto de origen se incrementa con cada paquete, así que si queremos utilizar **siempre el mismo puerto** utilizaremos la opción **--keep**.

El parámetro **--udp**, por supuesto, indica que el protocolo a utilizar será **UDP**.

El parámetro **--destport** es el **puerto de destino** del paquete.

El parámetro **--file** es el **fichero** en el que tenemos el campo **DATOS** del paquete, es decir, por ejemplo la consulta DNS, o la respuesta falsa para el caso de que estemos haciendo un ataque de envenenamiento de cache DNS.

El parámetro **--data** es el tamaño de los datos sin la cabecera, es decir, en este caso sería igual al campo **Tamaño paquete** de la cabecera UDP, pero **restándole 8 bytes**, que son los que ocupa la cabecera UDP.

Si quisiésemos especificar un **puerto de origen** usaríamos el parámetro **--baseport**. Si no se especifica, el puerto de origen será aleatorio.

Una opción curiosa de hping es la opción **--badcksum** que genera una **suma de comprobación inválida** en el paquete enviado, lo cual puede ser útil para comprobar la reacción de un sistema ante un paquete malformado.

A lo largo del curso, entraremos en más detalle en el funcionamiento de éstas y otras herramientas. Por el momento, os animo a que vayáis investigando por vuestra cuenta.

Autor: PyC (LCo)

RFC 768 J. Postel
ISI
28 de Agosto de 1980

PROTOCOLO DE DATAGRAMAS DE USUARIO
(User Datagram Protocol)

(Traducción al castellano: Diciembre de 1999)
(Por Domingo Sánchez Ruiz <domingo@quark.fis.ucm.es>)

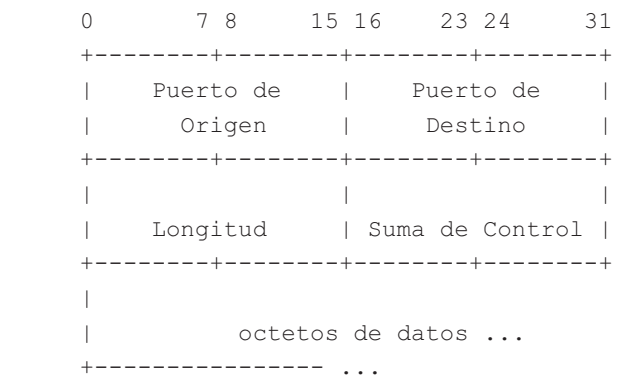
Introducción

Este Protocolo de Datagramas de Usuario (UDP: User Datagram Protocol)se define con la intención de hacer disponible un tipo de datagramas para la comunicación por intercambio de paquetes entre ordenadores en el entorno de un conjunto interconectado de redes de computadoras.

Este protocolo asume que el Protocolo de Internet (IP: Internet protocol) [1] se utiliza como protocolo subyacente.

Este protocolo aporta un procedimiento para que los programas de aplicación puedan enviar mensajes a otros programas con un mínimo de mecanismo de protocolo. El protocolo se orienta a transacciones, y tanto la entrega como la protección ante duplicados no se garantizan.
Las aplicaciones que requieran de una entrega fiable y ordenada de secuencias de datos deberían utilizar el Protocolo de Control de Transmisión (TCP: Transmission Control Protocol). [2]

Formato



Formato de la Cabecera de un Datagrama de Usuario

Campos

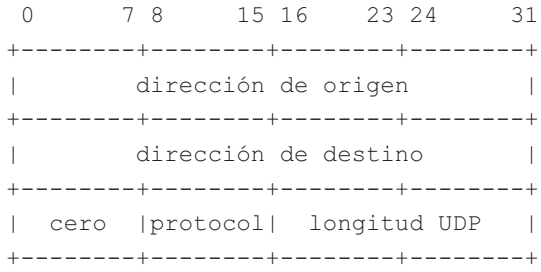
El campo Puerto de Origen es opcional; cuando tiene sentido, indica el puerto del proceso emisor, y puede que se asuma que ése sea el puerto al cual la respuesta debería ser dirigida en ausencia de otra información. Si no se utiliza, se inserta un valor cero.

El campo Puerto de Destino tiene significado dentro del contexto de una dirección de destino en un entorno internet particular.

El campo Longitud representa la longitud en octetos de este datagrama de usuario, incluyendo la cabecera y los datos. (Esto implica que el valor mínimo del campo Longitud es ocho.)

El campo Suma de Control (Checksum) es el complemento a uno de 16 bits de la suma de los complementos a uno de las palabras de la combinación de una pseudo-cabecera construida con información de la cabecera IP, la cabecera UDP y los datos, y rellenada con octetos de valor cero en la parte final (si es necesario) hasta tener un múltiplo de dos octetos.

La pseudo-cabecera que imaginariamente antecede a la cabecera UDP contiene la dirección de origen, la dirección de destino, el protocolo y la longitud UDP. Esta información proporciona protección frente a datagramas mal encaminados. Este procedimiento de comprobación es el mismo que el utilizado en TCP.



Si la suma de control calculada es cero, se transmite como un campo de unos (el equivalente en la aritmética del complemento a uno). Un valor de la suma de control trasmitido como un campo de ceros significa que el emisor no generó la suma de control (para depuración o para protocolos de más alto nivel a los que este campo les sea indiferente).

Interfaz de Usuario

Un interfaz de usuario debería permitir:

la creación de nuevos puertos de recepción, operaciones de recepción en los puertos de recepción que devuelvan los octetos de datos y una indicación del puerto de origen y de la dirección de origen, y una operación que permita enviar un datagrama, especificando los datos y los puertos de origen y de destino y las direcciones a las que se debe enviar.

Interfaz IP

El módulo UDP debe ser capaz de determinar las direcciones de origen y destino en un entorno internet así como el campo de protocolo de la cabecera del protocolo internet. Una posible interfaz UDP/IP devolvería el datagrama de internet completo, incluyendo toda la cabecera, en respuesta a una operación de recepción. Un interfaz de este tipo permitiría también al módulo UDP pasar un datagrama deinternet completo con cabecera al módulo IP para ser enviado. IP verificaría ciertos campos por consistencia y calcularía la suma de control de la cabecera del protocolo internet.

Aplicación del Protocolo

Los usos principales de este protocolo son el Servidor de Nombres de Internet [3] y la Transferencia Trivial de Ficheros (Trivial File Transfer) [4].

Número del protocolo

Este es el protocolo 17 (21 en octal) cuando se utilice en el Protocolo de Internet (IP). Se indican otros números de protocolo en [5].

Referencias

[1] Postel, J., "Internet Protocol," RFC 760, USC/Information Sciences Institute, Enero de 1980. (Nota del T. Hay traducción al español por P.J. Ponce de León: "Protocolo Internet", Mayo 1999.)

[2] Postel, J., "Transmission Control Protocol," RFC 761, USC/Information Sciences Institute, Enero de 1980.

[3] Postel, J., "Internet Name Server," USC/Information Sciences Institute, IEN 116, Agosto de 1979.

[4] Sollins, K., "The TFTP Protocol," Massachusetts Institute of Technology, IEN 133, Enero de 1980.

[5] Postel, J., "Assigned Numbers," USC/Information Sciences Institute, RFC 762, Enero de 1980.

Nota del traductor

Este documento y las traducciones al español mencionadas en las referencias pueden encontrarse en: http://lucas.hispalinux.es/htmls/estandares.html

El proyecto de traducción de RFC al español tiene su web de desarrollo en: http://www.arrakis.es/~pjleon/rfc-es