HACK X CRACK

Manipulación avanzada de paquetes TCP/IP con Scapy - Parte I

By Larry

hack crack
WWW.HACKXCRACK.ES

Manipulación avanzada de paquetes TCP/IP con Scapy – Parte I

En este artículo voy a introducirlos en el manejo de paquetes TCP/IP con una herramienta llamada Scapy. Como primer práctica vamos a realizar el hijacking o "robo" de una sesión TCP/IP ya establecida.

1 Introducción

Toda la Internet está basada en el contínuo intercambio de paquetes entre los distintos nodos de la red, con el objetivo de establecer comunicaciones. Al nivel más primitivo, estos paquetes son una serie de pulsos digitales bien definidos que viajan en cables de cobre alrededor del mundo. Además, la arquitectura de Internet es tal que todos estamos conectados con todos, con el objetivo de maximizar la eficiencia a la hora establecer comunicaciones. Es evidente que una red en la que hay que establecer nuevas conexiones físicas cada vez que se quiera utilizar algún servicio va a ser muchísimo más lenta que una red en la que está todo conectado de antemano.

Para evitar que la red sea un completo caos, nace el concepto de conexión virtual que permite que una comunicación entre dos nodos no sea escuchada por otras personas ajenas a la conversación. En otras palabras, con el protocolo TCP/IP de por medio, se puede aislar una conversación entre dos nodos de la red evitando que terceras personas ilegítimas participen en la conversación.

Pero como en este mundo nada funciona como debería, trabajando en un nivel adecuado se pueden modificar paquetes correspondientes a una conversación ajena y hasta inyectar datos maliciosos en dicha conversación. En este artículo se cubren técnicas que involucran la construcción y manipulación de paquetes para "controlar" la red y alterar su normal funcionamiento. Para eso, vamos a introducir una herramienta muy poderosa llamada Scapy. Antes de empezar, voy a listar los conocimientos básicos que debería tener un posible lector,

1. **Conocimiento sobre el protocolo TCP y UDP**: Básicamente, este artículo se trata de la manipulación de paquetes TCP/IP o UDP/IP para llevar a cabo un determinado objetivo. Por lo tanto, es esencial que el lector conozca los lineamientos básicos de estos protocolos de comunicación. De todas formas, a modo de repaso se explican la mayoría de los conceptos utilizados.

2. **Programación en general**: Como veremos,

Scapy está basado en el intérprete de Python y a lo largo del artículo se presentan algunos conceptos relacionados con dicho lenguaje. Además vamos a utilizar conceptos de programación como variables, enteros, cadenas de texto, etc. No se requieren conocimientos previos de Python, sin embargo supongo que cuando estés tan emocionado con el tema como lo estoy yo, querrás aprender inmediatamente. Personalmente, lo que más me gusta de Python es que puedes escribir cualquier cosa que crees que debería funcionar y FUNCIONA.

Finalmente, para motivarlos a que lean el artículo, enumero las cosas que podrían hacer usando Scapy:

1. Crear y manipular fácilmente paquetes TCP/IP.

2. Reproducir en muy pocas líneas con un grado de dificultad muy bajo, herramientas como: traceroute, tcptraceroute, arpspoof, dnsspoof, algún escaner de puertos, etc. El objetivo de esto no es "reinventar la pólvora" sino comprender a fondo el funcionamiento de dichas aplicaciones, que muchos de nosotros en algún momento las utilizamos como "cajas negras" y personalizarlas para llevar a cabo tareas muy particulares.

3. Crear gráficos 2D y 3D de traceroutes.

4. Reproducir en muy pocas líneas un snnifer y modificar paquetes TCP/IP en tiempo real.

5. Scapy es una herramienta muy fácil de usar que puede ayudar a una persona a comprender a fondo el protocolo TCP/IP y otros protocolos a nivel de aplicación.

6. Infinitas cosas más, limitadas sólo por la imaginación y el grado de conocimiento del usuario.

Para descargar Scapy, en la página web http://www.secdev.org/projects/scapy hay bastante información sobre la instalación de la versión más reciente. En la mayoría de los sistema operativos basados en Debian, simplemente basta hacer "sudo apt-get install scapy". La herramienta está pensada para entornos de tipo Unix, pero también se la puede instalar en Windows.

2 ¿Porqué crear paquetes personalizados?

Un paquete es una unidad de información bien ordenada que se encuentra agrupada con una determinada estructura (son unos y ceros que tienen algún "significado"). En cierta forma, comparte los atributos de una carta que se envía por correo postal. Contiene información sobre el origen y el destino del paquete, la prioridad del paquete, la ruta que debe seguir para llegar a su destino y, por supuesto, el contenido que debe ser entregado al destinatario.

En resumen, un paquete (independientemente del protocolo utilizado) contiene los datos necesarios para que pueda ser transmitido correctamente y el "payload", que es el contenido que debe ser entregado. Toda esta información es procesada por funciones automatizadas que se adhieren a estrictas reglas que son conocidas públicamente. Aquí es donde la creación de paquetes personalizados juega un rol importante: no importa donde ni como se generó un paquete, si este se adhiere a las normas preestablecidas, una máquina lo va a procesar si recibe lo que estaba esperando. Para visualizar esto, veamos dos ejemplos que además nos van a ayudar a familizarizarnos con Scapy y con su línea de comandos interactiva.

2.1 Creación de un paquete UDP

El escenario en este simple ejemplo se muestra en la figura 1. Hay dos máquinas, una con IP 192.168.0.100 (máquina A) y otra con IP 192.168.0.106 (máquina B) que tiene el puerto 1024 UDP abierto y a la escucha (LISTENING) con el netcat (con la opción "-u" , que indica modo UDP).

Por lo tanto, la máquina 192.168.0.106 está esperando un paquete UDP por el puerto 1024. A continuación, vamos a armar un paquete UDP arbitrario en la máquina A con Scapy y se lo mandamos a la máquina B. En la figura 2 se muestran los comandos ejecutados que a continuación pasamos a explicar.

Básicamente, en Scapy se contruye un paquete por capas. Es decir, por un lado construimos la capa IP con los parámetros que deseamos (dirección IP de destino y de origen, Time to Live, Protocolo, etc.) y por otro lado construimos la capa UDP con los puertos de destino y origen y la capa de aplicación donde está el "payload" o los datos que queremos enviar. También se puede personalizar la capa Ethernet si se desea trabajar a niveles bajos.

Una vez que tenemos todas las capas armadas, las "juntamos" con el operador "/" armando el paquete final a ser enviado. Finalmente, enviamos el paquete con el comando send() y automáticamente se completan todos los campos que no modificamos con valores por defecto del sistema operativo o que dependen del paquete (el checksum, por ejemplo). Bastante simple, ¿no?. A continuación vamos a explicar los comandos que se muestran en la figura 2:

>ls(IP()): Este comando lista todos los campos de la clase IP() que pueden ser personalizados. El comando ls(IP()) nos muestra cuatros columnas: la primera el nombre de cada campo, la segunda el tipo de dato de cada campo, la tercera los valores por defecto al iniciar Scapy y la última son los valores actuales de cada uno de los atributos (como no modificamos nada todavía, son iguales a los de la tercera columna).

>capa_IP=IP(dst="192.168.0.106"): Este comando hace dos cosas y voy a tratar de explicarlas sin usar un lenguaje específico de Python. Primero establece un valor de 192.168.0.106 al campo dst de la clase IP(). Segundo "iguala" la "variable" capa_IP a la clase IP() con este valor modificado. Es decir, ahora capa_IP tiene las mismas propiedades que la clase IP() (IP de origen, IP de destino, TTL, etc) pero con el campo dst igual a 192.168.0.106, que evidentemente representa la IP de destino. Por ejemplo, si ahora ejecutamos el comando ls(capa_IP), obtendremos una salida parecida al ítem anterior, sólo que en la cuarta columna en la fila correspondiente el campo dst aparece la IP que seteamos nosotros.

>capa_IP: Este comando muestra en pantalla los campos que fueron modificados por el usuario (distintos a los que vienen por default).

>ls(UDP()): Este comando lista los distintos atributos de la clase UDP.

>capa_UDP=UDP(dport=1024 , sport=5025): Este comando modifica los valores por defecto de la clase UDP y "setea" los puertos de origen y destino al 5025 y 1024 respectivamente. El puerto 1024 es en el que está escuchando la máquina 192.168.0.106 y el puerto de origen 5025 es un número cualquiera de 16 bits.

>capa_UDP: De vuelta, se muestra en pantalla los campos que fueron modificados por el usuario (distintos a los que vienen por default).

>payload="Mandamos un paquete UDP :-)\n":

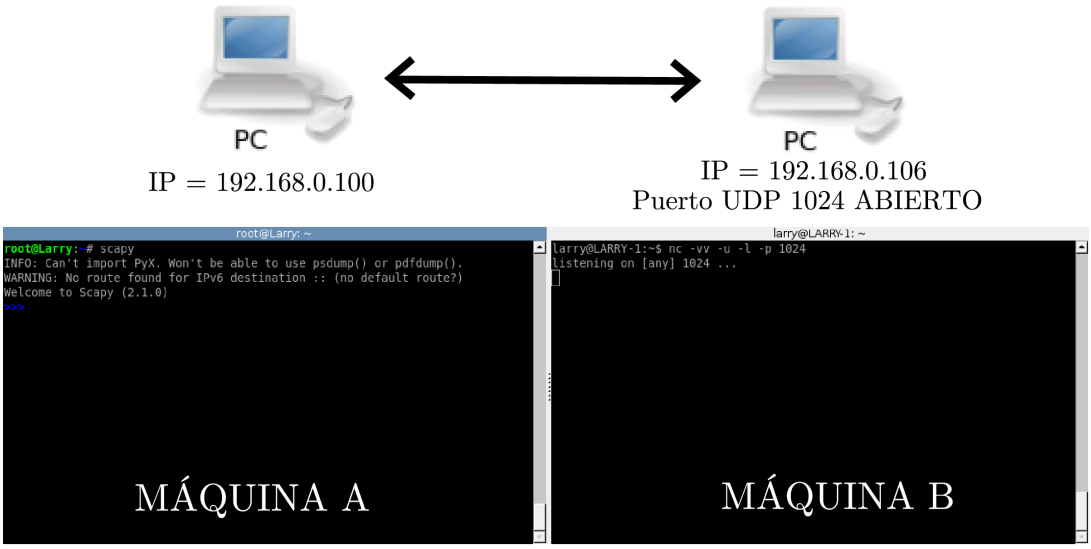


Figura 1. Escenario simple en una red interna. A la izquierda se encuentra la máquina A que inició una sesión en Scapy y a la derecha se encuentra la máquina B, que tiene el netcat a la escucha en el puerto 1024.

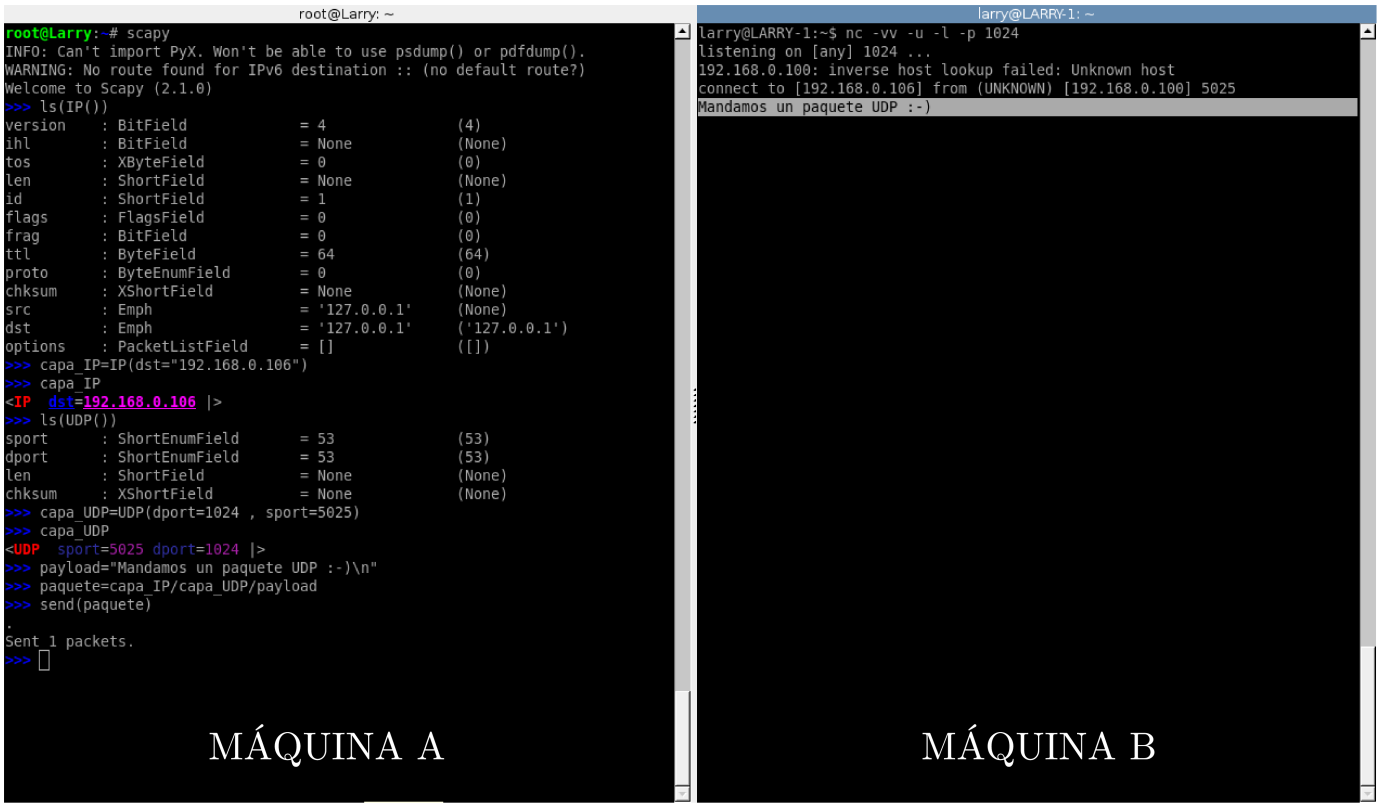


Figura 2. Sesión interactiva en Scapy para la creación de un paquete UDP.

Acá simplemente inicializamos una variable de tipo "string" que representa el payload que va adjunto al paquete UDP/IP. En otras palabras, representa la capa de aplicación donde se incluyen los comandos que debería entender la aplicación que está escuchando en el puerto UDP.

>paquete=capa_IP/capa_UDP/payload: Finalmente juntamos las tres capas utilizando el operador "/" y asi armamos un paquete que tiene toda la información que ingresamos anteriormente. Si hacemos ls(paquete) se imprime en pantalla los valores de los campos de cada capa del paquete.

>send(paquete): Una vez armado el paquete, lo enviamos con el comando send(). Los campos que no completamos en la capa IP y UDP (checksum, versión, protocolo de la capa anterior, longitud, etc.) se "llenan" automáticamente con valores que pueden depender del paquete o del sistema operativo. Además, en ningún momento modificamos ni armamos una capa Ethernet pero igual el paquete se pudo enviar correctamente a la máquina 192.168.0.106 (como puede verse a la derecha de la figura 2). Esto se debe a que la función send() trabaja en la capa 3 (capa IP). Si quieren personalizar la capa de enlance hay que usar la función sendp(), que trabaja en la capa 2.

Como vemos, Scapy es una aplicación muy versátil y fácil de utilizar. A continuación, complicamos un poco más el escenario para demostrar el alcance de esta poderosa herramienta.

2.2 Inyección de paquetes UDP en una conversación externa

En este nuevo escenario, que se muestra en la figura 3 hay tres máquinas en una red interna: la máquina A con IP 192.168.0.100, la máquina B con IP 192.168.0.106 con el puerto UDP 1024 a la escucha y la máquina C con IP 192.168.0.107 que intercambia paquetes UDP con la B utilizando el puerto 49392 de origen. Además, la maquina C envió un paquete UDP a la máquina B con un payload que contiene la cadena: "HOLA 106, soy 107".

Como se muestra en la figura 3, la máquina A de alguna forma comprometió el segmento de red entre B y C y tiene conocimiento de todos los detalles de la conversación UDP que se está llevando a cabo. Es decir, sabe que la IP 192.168.0.107 envía paquetes UDP del puerto 49392 al puerto 1024 de la máquina con IP 192.168.0.106 (esto se puede lograr mediante la técnica de ARPSpoofing, la cual explicaremos más adelante).

En resumen, el atacante en la máquina A sabe

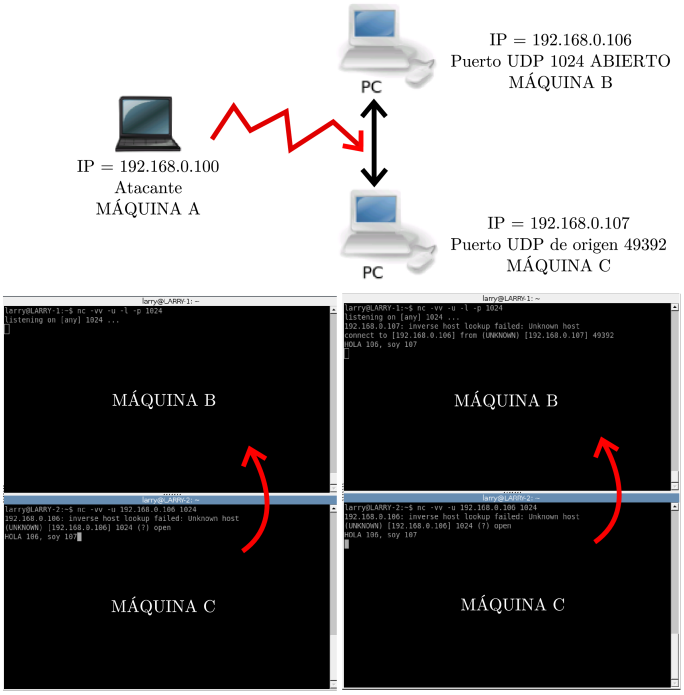


Figura 3.Escenario simple donde la máquina B intercambia paquetes UDP con la máquina C. El atacante, en la máquina A, logra comprometer el segmento de red entre B y C e inyecta paquetes ilegítimos en la conversación ``UDP''.

que la máquina B va a recibir y procesar correctamente paquetes UDP que provengan de la IP 192.168.0.107 con puerto de origen 49392, independientemente de como y donde se creen. Los únicos cuatro parámetros que definen unívocamente la conversación entre B y C son:

1. IP de la máquina B: 192.168.0.106

2. IP de la máquina C: 192.168.0.107

3. Puerto UDP de la máquina B: 1024

4. Puerto UDP de la máquina C: 49392

Con algún objetivo malicioso, el atacante en la máquina A busca suplantar la identidad de la máquina C inyectando paquetes ilegítimos en la conversación. Para eso, lleva a cabo una sesión interactiva en Scapy que se muestra en la figura 4, e inyecta un paquete UDP que tiene como payload la cadena "HOLA 106, soy 100 pero me estoy haciendo pasar por 107\n". A continuación, pasamos a detallar los comandos ejecutados para realizar dicha acción:

>capa_IP=IP(src="192.168.0.107", dst="192.168.0.106"): De forma análoga al caso anterior, establecemos la IP de origen del paquete como la 192.168.0.107 (máquina C, que es la que el atacante quiere suplantar) y la IP de destino como 192.168.0.106 (máquina B).

>capa_IP: Imprimimos en pantalla los cambios que realizamos sobre los valores por defecto.

>capa_UDP=UDP(sport=49392, dport=1024): Configuramos los puertos de origen y destino del paquete.

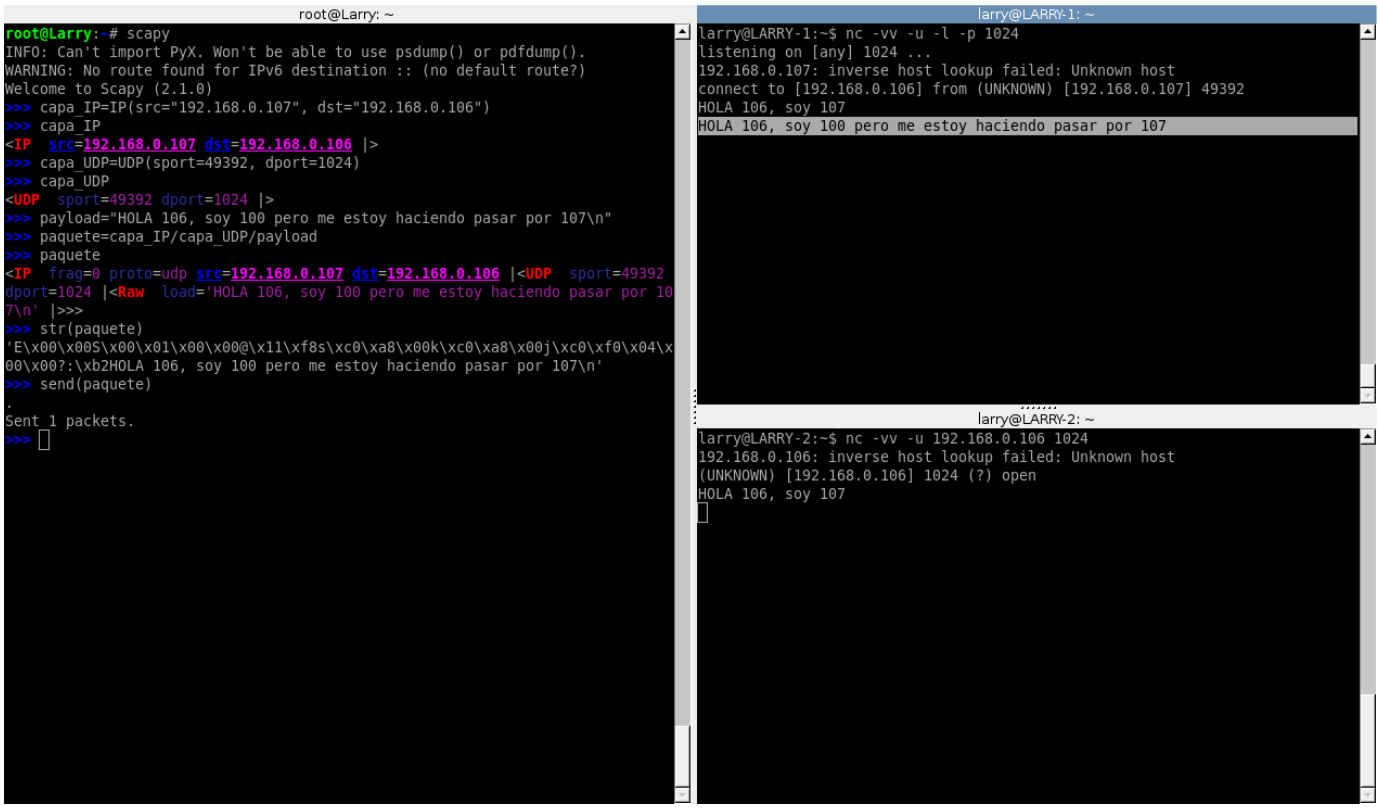


Figura 4. Sesión interactiva en Scapy, donde el atacante (con IP 192.168.0.100) suplanta la identidad de la máquina C (con IP 192.168.0.107) en una "conversación'' UDP enviando un paquete personalizado con Scapy.

>capa_UDP: De vuelta, imprimimos los cambios realizados para controlar que todo esté bien.

>payload="HOLA 106, soy 100 pero me estoy haciendo pasar por 107\n": Inicializamos una cadena de caracteres que se utilizará como payload del paquete UDP.

>paquete=capa_IP/capa_UDP/payload: Finalmente armamos el paquete que vamos a enviar.

>paquete: Verificamos el paquete por última vez, para asegurarnos de que todos los parámetros "seteados" sean correctos.

>str(paquete): Realiza una "disección" del paquete y lo imprime en pantalla como una serie de caracteres hexadecimales.

>send(paquete): Enviamos el paquete y, como se puede observar en la consola superior derecha de la figura 4, llegó a la máquina B.

El escenario anterior muestra con que facilidad podemos suplantar la identidad de un host en una "conversación" de tipo UDP. Aunque el ejemplo anterior parezca un poco tonto y carente de sentido no está muy lejos de ser, por ejemplo, una respuesta DNS de un servidor de nombres de dominio a un determinado cliente. El caso anterior generalmente se da cuando se intercambian datos utilizando un protocolo no orientado a conexión (UDP). En ese caso, la capa de transporte no tiene implementado ni un mecanismo que identifique el flujo de información y que evite que terceros "inyecten" paquetes deliberadamente en la conversación (aunque se podría implementar uno en la capa de aplicación, como el caso del protocolo DNS).

\\\\\\\\\\\\\\\\\\\\\\\\\----------=

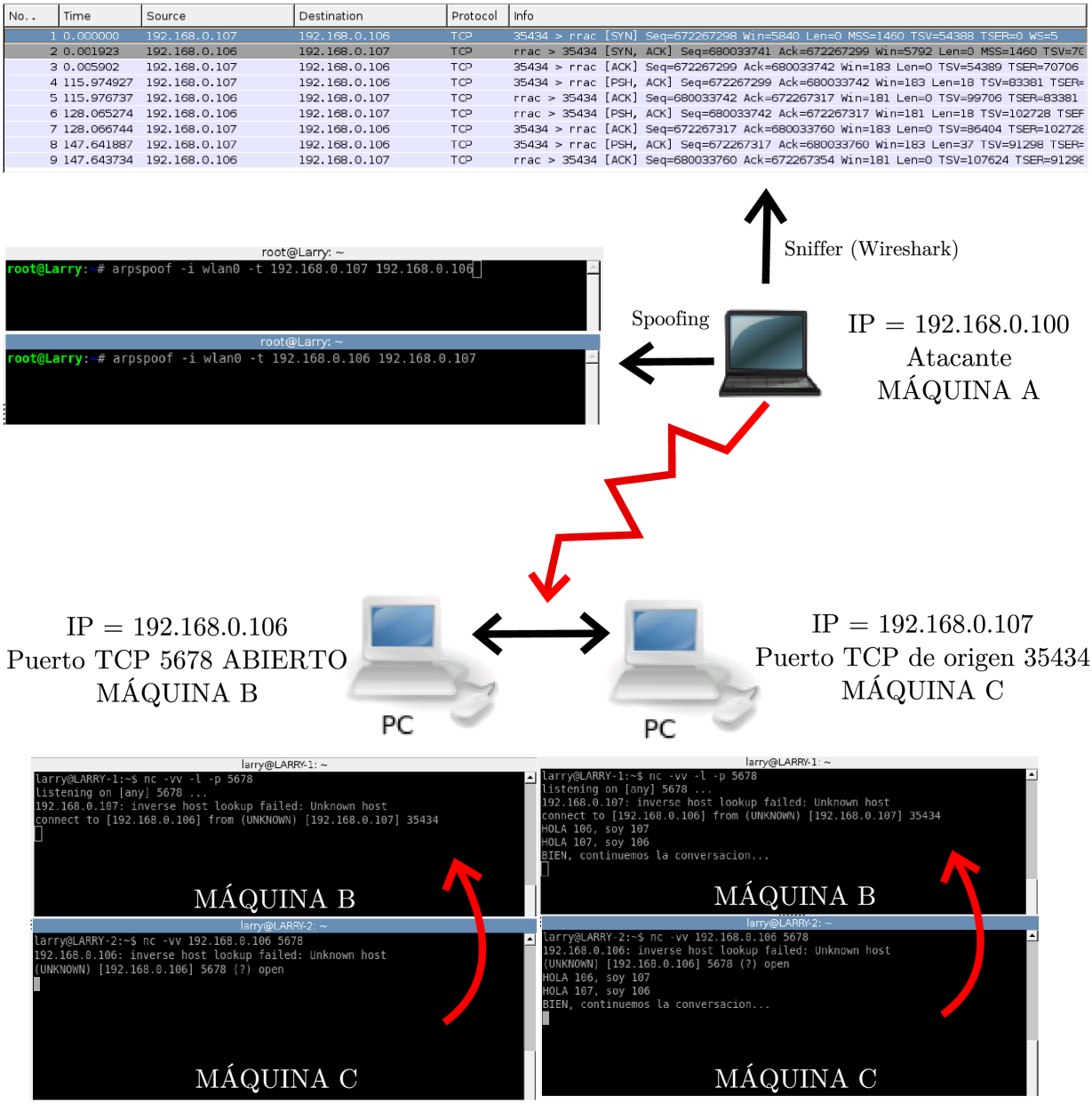
3 Hijacking de una sesión TCP/IP

Complicamos aún más el escenario anterior. En vista de lo que sucedió anteriormente en la conversación UDP, las máquinas B y C deciden que es más conveniente intercambiar los paquetes mediante una conexión TCP. Como sabemos, el protocolo TCP es orientado a conexión e implementa un mecanismo en la capa de transporte para el control del flujo de información y la creación de un par de circuitos virtuales (cada uno en una dirección). Esto permite que sólo los dos sistemas finales sincronizados puedan usar la conexión (en este caso, B y C).

Las conexiones TCP se componen de tres etapas: establecimiento de conexión, transferencia de datos y fin de la conexión. Para establecer la conexión se usa el procedimiento llamado negociación en tres pasos (3-way handshake) y para la desconexión se utiliza una negociación en cuatro pasos (4-way handshake). Durante el establecimiento de la conexión, algunos parámetros como el número de secuencia (SEQ) son configurados para asegurar la entrega ordenada de los datos y la robustez de la comunicación.

El escenario de este ejemplo es similar al anterior y se muestra en la figura 5. Hay una máquina B con IP 192.168.0.106 con el puerto TCP 5678 abierto y una máquina C con IP 192.168.0 107 que utiliza el puerto 35434 de origen para establecer una conexión TCP con la máquina B. El atacante en la máquina A con IP 192.168.0.100 comprometió el segmento de red entre B y C utilizando la conocida (y poco sigilosa) técnica de ARPSpoofing, ejecutando los comandos que se muestran en la figura 5.

El principio del ARPSpoofing es enviar mensajes ARP falsos (falsificados, o spoofed) a la Ethernet. La finalidad de esta técnica es asociar la



RESUMEN de la conversación (resumida del sniffer)

3-Way Handshake

IP 192.168.0.107 →	**SYN**, Seq=672267298	→ IP 192.168.0.106
IP 192.168.0.107 ←	**SYN/ACK**, Seq=680033741, Ack=672267299	← IP 192.168.0.106
IP 192.168.0.107 →	**ACK**, Seq=672267299, Ack=680033742	→ IP 192.168.0.106

Transferencia de datos

IP 192.168.0.107 →	**PSH/ACK**, "HOLA 106, soy 107"	→ IP 192.168.0.106
IP 192.168.0.107 ←	**ACK**, Seq=672267299, Ack=680033742	← IP 192.168.0.106
IP 192.168.0.107 ←	**PSH/ACK**, "HOLA 107, soy 106"	← IP 192.168.0.106
IP 192.168.0.107 →	**ACK**, Seq=672267317, Ack=680033760	→ IP 192.168.0.106
IP 192.168.0.107 →	**PSH/ACK**, "BIEN, continuemos la conv..."	→ IP 192.168.0.106
IP 192.168.0.107 ←	**ACK**, Seq=680033760, Ack=672267354	← IP 192.168.0.106

Figura 5. Escenario donde un atacante A (IP 192.168.0.100) se dispone al robo de una sesión TCP/IP entre los hosts B y C.

dirección MAC del atacante con la dirección IP de otro nodo (el nodo atacado). En este caso particular, el atacante hace creer a B que la IP 192.168.0.107 tiene asociada su MAC y a la máquina C que la IP 192.168.0.106 también tiene asociada su MAC. En conclusión, todos los paquetes que intercambian B y C pasan físicamente por la máquina A.

El escenario completo se muestra en la figura 5, donde se observa que el atacante fue capaz de captar el tráfico intercambiado entre B y C (que también se muestra en dicha figura), y obtener todos los números de secuencia y de acuse de recibo de la conversación. De vuelta, con algún objetivo malicioso, el atacante en la máquina A busca suplantar la identidad de la máquina C inyectando paquetes ilegítimos en la conversación. Para eso, lleva a cabo la sesión interactiva en Scapy que se muestra en la figura 6.

Este caso es un poco más complicado que el anterior (UDP) pero no muy difícil de llevar a cabo. Simplemente hay que armar un paquete PSH/ACK con los números de secuencia y de acuse de recibo que espera la máquina B e inyectarlo de forma análoga al caso de UDP. Como el atacante pudo captar la conversación entre B y C sabe que números de SEQ y ACK debe tener el paquete y se dispone a armarlo en Scapy, usando los siguientes comandos (que se muestran en la figura 6):

>capa_IP=IP(src="192.168.0.107", dst="192.168.0.106"): De forma análoga a los casos anteriores, establecemos la IP de origen del paquete como la 192.168.0.107 (máquina C)

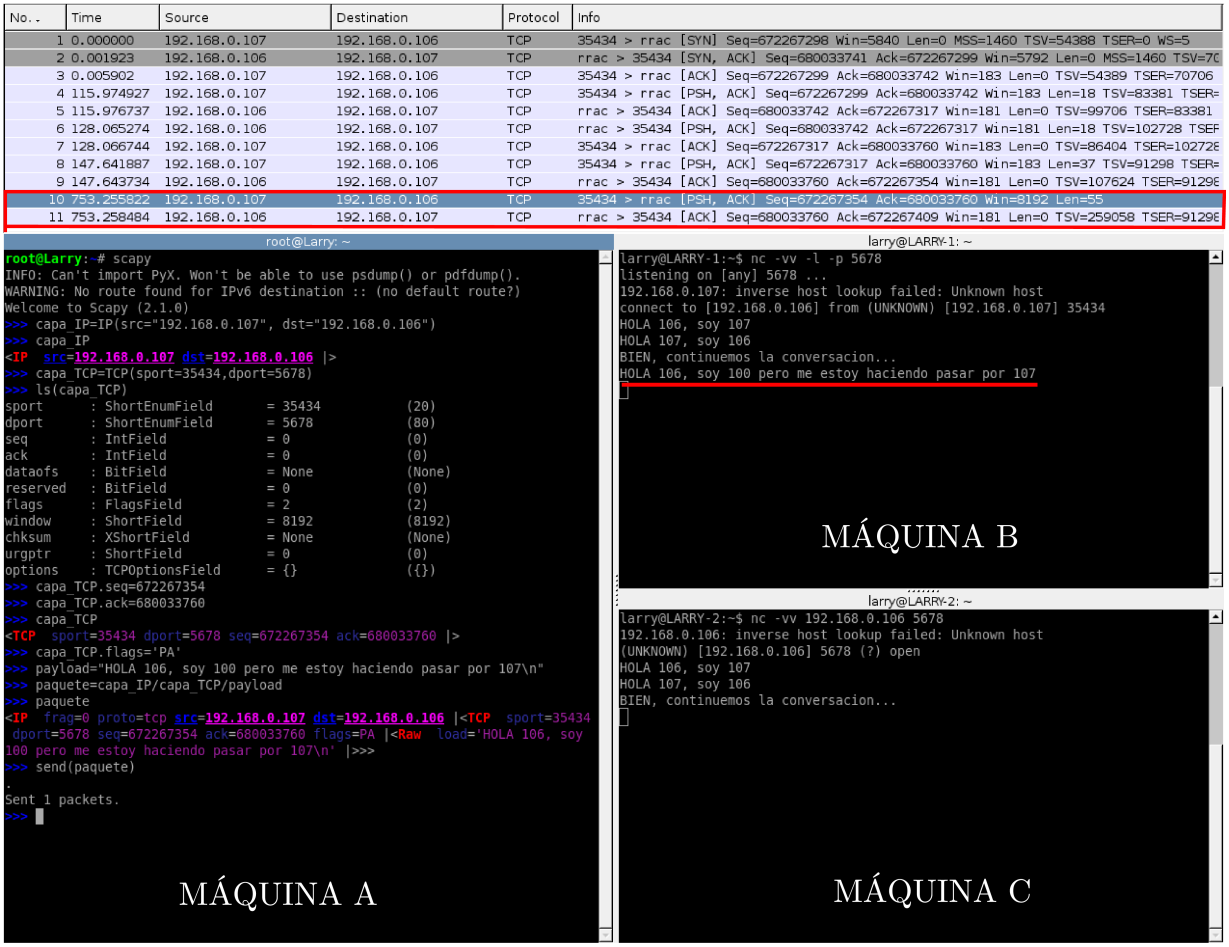


Figura 6. Inyección de un paquete TCP en una conversación ajena.

y la IP de destino como 192.168.0.106 (máquina B).

>capa_IP: Imprimimos en pantalla los cambios que realizamos sobre los valores por defecto.

>capa_TCP=TCP(sport=35434, dport=5678): Al igual que en el caso de un paquete UDP, configuramos los puertos de origen y destino del paquete.

>ls(capa_TCP): Imprimimos en pantala los campos de la clase capa_TCP que podemos modificar (que obviamente son todos los de la capa TCP). En particular, observamos que hay dos campos que hacen referencia al número de secuencia y de acuse de recibo: seq y ack, respectivamente.

>capa_TCP.seq=672267354: Este comando modifica el campo seq de la clase capa_TCP con el valor que nosotros queramos, que en este caso es el 672267354. Si observan detalladamente la figura 5, el último paquete que la máquina B envió a la máquina C es un paquete ACK, con número de secuencia igual a 680033760 y número de acuse de recibo 672267354. Por lo tanto, el próximo paquete PSH/ACK que espera la máquina B de la máquina C debe tener un número de secuencia igual a 672267354 y un número de acuse de recibo igual a 680033760 ("intercambiamos" los números).

>capa_TCP.ack=680033760: Este comando modifica el campo ack de la clase capa_TCP con el valor que nosotros queramos, que en este caso es el 680033760.

>capa_TCP: Este comando imprime en pantalla los valores de los campos que modificamos.

>capa_TCP.flags='PA': Con este comando se indica que el paquete debe tener los flags PSH y ACK levantados.

>payload="HOLA 106, soy 100 pero me estoy haciendo pasar por 107\n": Inicializamos una cadena de caracteres que se utilizará como payload del paquete UDP.

>paquete=capa_IP/capa_TCP/payload: Armamos el paquete que vamos a enviar.

>paquete: Verificamos el paquete por última vez, para asegurarnos de que todos los parámetros "seteados" sean correctos.

>send(paquete): Enviamos el paquete y, como se puede observar en la consola superior derecha de la figura 6, llegó a la máquina B.

Como vemos, el atacante fue capaz de suplantar la identidad de la máquina C en la conexión TCP que ya estaba establecida. Es más, si la IP del atacante (192.168.0.100) fuera una IP NO AUTORIZADA a comunicarse con B, éste método igual hubiera funcionado, ya que suplantamos la IP del cliente legítimo (IP Spoofing). La pregunta es, ¿qué pasa con la sesión legítima iniciada en la máquina C? ¿Qué pasa si el cliente en la maquina C desea enviar algún dato a la máquina B? La respuesta es que no pasa absolutamente nada. Sus números de secuencia y acuse de recibo van a estar totalmente "desfasados" en la conexión y la máquina B (que se adhiere estrictamente al protocolo TCP) va a rechazar cada uno de los paquetes que envíe C.

El atacante ahora desea deshacerse del cliente en la máquina C, para lo cual lleva a cabo la sesión en Scapy que se muestra en la figura 7. Básicamente, arma un paquete TCP con el flag "RESET" para suplantar a la máquina B y enviárselo a la máquina C. Como el atacante tiene conocimiento de los números SEQ y ACK que espera la máquina C de la máquina B, lo puede armar convenientemente adhiriéndose al

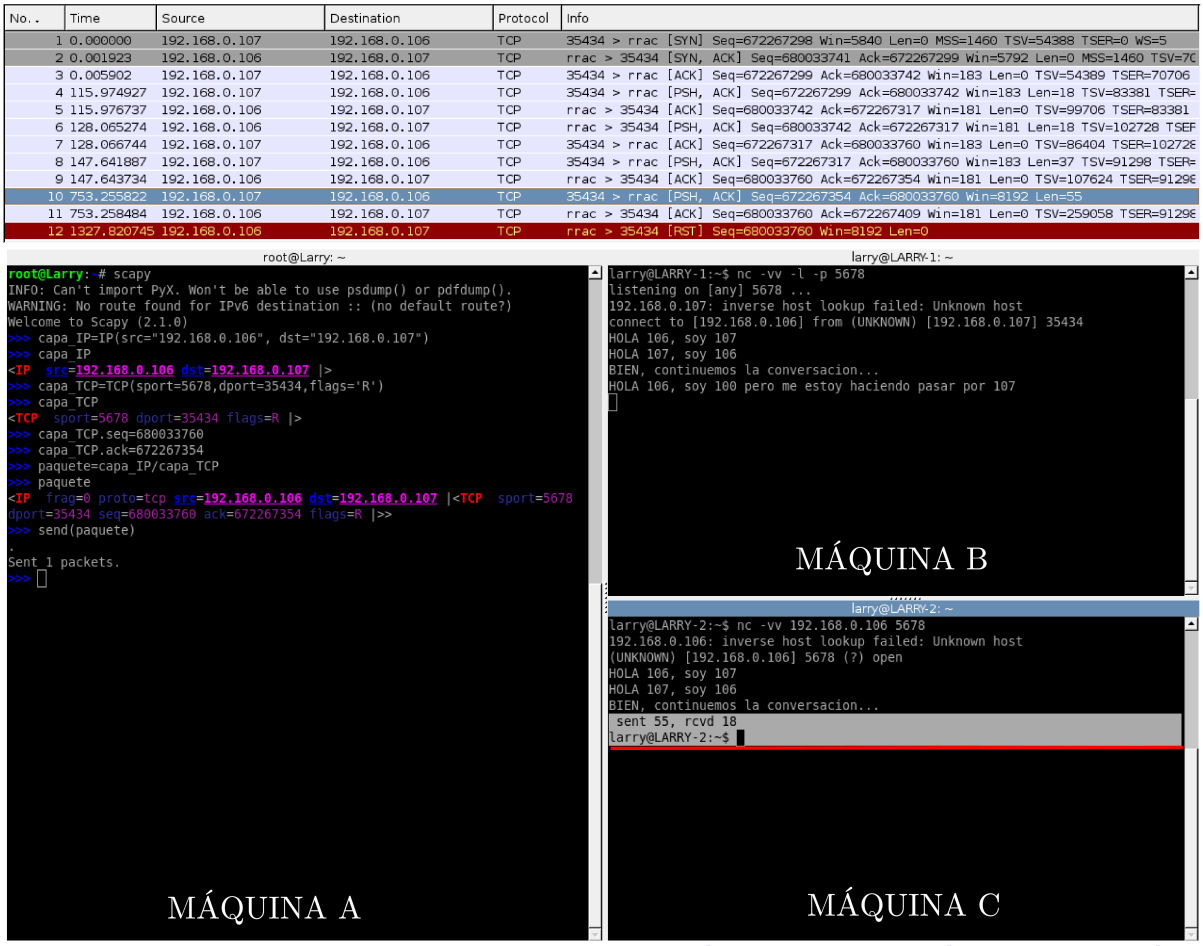


Figura 7. El atacante crea un paquete personalizado para terminar la conexión de la máquina C con la B.

protocolo TCP (los números SEQ y ACK se muestran en la figura 5).

Como se puede ver en la figura 8, la sesión de netcat en la máquina C se cierra automáticamente luego de que A envía el paquete RESET. La explicación de los comandos utilizados es análoga al caso anterior, con la única diferencia de que ahora el campo flags de la clase capa_TCP es 'R'. Con respecto al número de secuencia utilizado, antes de que la máquina A inyectara el paquete ilegítimo en la conversación, el último paquete que la máquina B envió a la máquina C es un paquete ACK, con número de secuencia igual a 680033760. Por lo tanto, este debe ser el número de secuencia del paquete RESET que tiene que crear el atacante.

Una vez que el atacante dejó fuera del escenario a la máquina C, puede llevar a cabo un ataque de denegación de servicio y dejar de redireccionar los paquetes de C a la máquina B (pero sin dejar de hacer ARPSpoofing). Esto se puede lograr simplemente haciendo “echo 0 > /proc/sys/net/ipv4/ip_forward”. Una técnica un poco más sutil sería usar IPTABLES para evitar que la máquina C vuelva a establecer una conexión con el puerto 5678 de la máquina B. De cualquier forma, si la máquina C lograra establecer nuevamente una conexión con B, no pasaría nada con la sesión robada por la máquina A ya que C utilizaría otro puerto de origen y otros números de secuencia.

Lo que si es esencial tener en cuenta es que, una vez robada la sesión, el atacante A debe dejar de

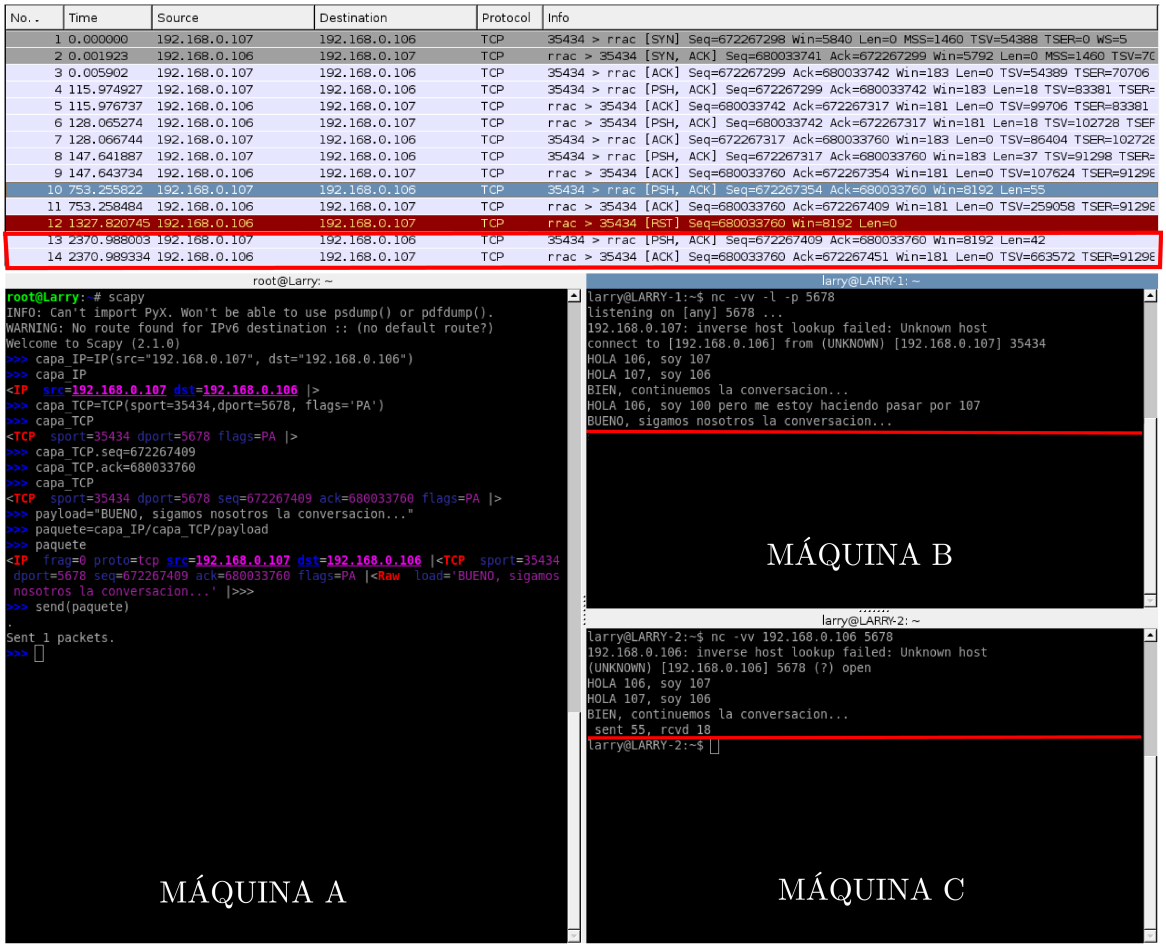


Figura 8. El atacante en la máquina A logra suplantar la identidad de la máquina C continuando la conversación TCP con la máquina B.

redireccionar los paquetes de la máquina B a la C. En caso contrario, seguirían llegando paquetes a C de una conexión que ya fue cerrada lo cual es SOSPECHOSO y también provocaría una ráfaga de paquetes RESET de la máquina C a la B, que podrían causar que el atacante pierda la conexión robada. Finalmente, en la figura 8 se muestra una sesión de Scapy donde el atacante continua la "conversación" con la máquina B suplantando la identidad de la máquina C.

Para concluir, notamos que todo lo expuesto en este artículo tiene solamente un interés académico. Los parámetros y las variables a controlar en un proceso de hijacking de una sesión TCP/IP son muchos y hacen que sea imposible aplicarlo en una situación real (sesiones HTTP, FTP, POP3, SMTP, etc) de una forma tan rústica como lo hicimos nosotros. Es evidente que hay que "automatizar" las distintas tareas y coordinar adecuadamente el ARPSpoofing, el monitoreo del tráfico y la inyección de paquetes.

Afortunadamente, la aplicación Scapy junto con el entorno de programación de Python hacen posible la construcción de una herramienta práctica de hijacking. Scapy puede trabajar en la capa de enlace realizando ARPSpoofing, puede monitorear los paquetes que pasan a través de nuestra máquina y obtener los números de secuencia y de acuse de recibo para "colarse" en una conexión TCP/IP de terceros. Todas esas tareas concurrentes se pueden "administrar" correctamente usando la programación de

threadings en Python. Para no alargar más este artículo, dejamos todos esos temas para otra ocasión.

4 Conclusión

La conclusión (bastante obvia) que podemos obtener de esta práctica es que las computadoras son ESTÚPIDAS. Nosotros somo seres INTELIGENTES y podemos aprovecharnos de ellas. Nunca perdamos de vista que los nodos de una red se comunican con “unos” y “ceros” que viajan a través del “ciberespacio” en estructuras bien definidas (y conocidas públicamente) llamadas PAQUETES. Una computadora va a procesar los “unos” y “ceros” que reciba si tienen algún sentido, independientemente de donde se armaron y de como llegaron. Por lo tanto, si una computadora recibe un paquete que estaba esperando, lo va a tratar correctamente acatando las normas o reglas con las cuales fue programada.